НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

"На правах рукопису"
УДК_____

«До захисту допущено»
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
"___"_____2018р.

Магістерська дисертація

зі спеціальності 122 Комп'ютерні науки та інформаційні технології
за спеціалізацією Програмне забезпечення розподілених систем
на тему " Система керування базою програмного коду на платформі Office 365"

Виконав: студент 6 курсу, групи _____
Литвиненко Дмитро Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н, доцент Тихоход В.О. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018

Національний технічний університет України
“ Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий, магістерський

зі спеціальності - 122 Комп’ютерні науки та інформаційні технології
за спеціалізацією - Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)
«____» _____ 2018р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Литвиненко Дмитро Сергійович
(прізвище, ім’я, по батькові)

1. Тема дисертації Система керування базою програмного коду на платформі Office 365

Науковий керівник Тихоход Володимир Олександрович, ктн, доц.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “__” _____ 2017 року №

2. Строк подання студентом дисертації 11 грудня 2018

3. Об’єкт дослідження проблеми організації взаємодії при розробці програмного продукту великою командою розробників

4. Предмет дослідження комп’ютерні технології систем керування версіями програмного коду та організації командної роботи

5. Перелік питань, які потрібно розробити проаналізувати існуючі методи контролю виконання задач, дослідити можливості аналогічних продуктів, проаналізувати можливості систем керування версіями програмного коду, розробити програмний продукт у вигляді компоненту до існуючої системи, дослідити правильність роботи розробленого модулю.

6. Орієнтований перелік ілюстративного матеріалу функціональні можливості користувачів, схема архітектури системи, схема відстежування змін, організація компонентів, структура програмного забезпечення, концептуальна модель предметної області, концептуальна модель даних систем контролю версій, архітектура MVVM, структурна схема навігації між сторінками додатку, інтерфейс користувача, методика роботи користувача з системою

7. Орієнтований перелік публікацій 1. Матеріали XVI Міжнародної науково-практичної конференції аспірантів, магістрантів і студентів «Сучасні проблеми наукового забезпечення енергетики» м. Київ, 24-27 квітня 2018 року, «Система керування базою програмного коду на платформі Office 365» _____
_____

8. Дата видачі завдання «__8__»_____грудня_____2017р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	строки виконання етапів магістерської дисертації	Примітка
1	Визначення теми та обговорення концепції магістерської роботи	9.10.17р.	
4	Опрацювання літературних джерел	29.10.17р.-10.12.17р.	
5	Проведення педагогічної практики на кафедрі	29.10.17р.-10.12.17р.	
6	Підготовка матеріалів магістерської роботи	05.02-11.05.18р.	
7	Проміжний контроль підготовки	02.04.18р.-06.04.18р.	
8	Підготовка доповідей на конференціях за темою магістерської роботи	05.02.18р.-31.03.18р.	
9	Доповідь на конференції	04.18р.	
10	Переддипломна практика	03.09.18р.-28.10.18р.	
11	Захист програмного продукту	22.10.18р.-25.10.18р.	
12	Розроблення стартап-проекту	19.11.18р.-31.11.18р.	
13	Передзахист роботи	26.11.18р.-30.11.18р.	
14	Оформлення диплома	03.12.18р.-10.12.18р.	
15	Здача всіх матеріалів на підпис зав. кафедрою	11.12.18р.	
16	Захист магістерських робіт	17.12.18р.	

Студент _____ ___Литвиненко_Д. С._____
(підпис) (прізвище та ініціали)

Науковий керівник _____ ____Тихоход В. О._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Структура та обсяг дипломної роботи

Магістерська дисертація складається зі вступу, п'яти розділів, висновку, переліку посилань з 52 найменувань, 2 додатків, і містить 40 рисунків, 15 таблиць. Повний обсяг магістерської дисертації складає 97 сторінок, з яких перелік посилань займає 5 сторінок, додатки – 4 сторінки.

Актуальність теми. Розробка складних інформаційних систем вимагає узгодженої роботи цілої групи програмістів. Проблеми організації взаємодії при розробці програмного продукту великою командою розробників зазвичай вирішується використанням системи керування версіями програмного коду (Version Control System, VCS) [1]. Така система дозволяє керувати поступовими змінами внесеними в електронні документи та відміняти ці зміни у разі необхідності.

В той же час ця система керування версіями не існує сама по собі, а тісно пов'язана із іншими внутрішніми інструментами та системами, що використовуються в процесі розробки та підтримки програмних продуктів. До них відносять системи документації, керування задачами (такими як Redmine, Jira), системами неперервної інтеграції (Jenkins) та внутрішніми продуктами компанії.

Наразі в системі відсутня можливість інтеграції із віддаленими репозиторіями систем керування версій. Саме тому розробка системи керування базою програмного коду на платформі Office 365 є важливим напрямком роботи. Це зумовлює актуальність розробки нових програмних засобів, інструментів моделювання та прогнозування тому, що значно полегшить та покращить якість взаємодії у команді та допоможе зберегти багато грошових ресурсів.

Мета дослідження полягає в розробці автоматизованої системи систему керування базою програмного коду.

Для досягнення поставленої задачі були сформульовані наступні **завдання дослідження**, що визначили логіку дослідження та його структуру:

- проаналізувати існуючі методи контролю виконання задач;
- дослідити можливості аналогічних продуктів;
- проаналізувати системи керування версіями програмного коду;
- розробити програмний продукт у вигляді компоненту до існуючої системи.

Об’єктом дослідження є проблеми організації взаємодії при розробці програмного продукту великою командою розробників.

Предметом дослідження є комп’ютерні технології керування версіями програмного коду та організації командної роботи.

Практичне значення одержаних результатів роботи полягає в розробці програмної платформи, яка б інтегрувалася в програмну систему Microsoft Sharepoint, та надавала можливості пов'язування усього вбудованого функціоналу системи із віддаленим репозиторієм системи контролю версій.

Апробація результатів дисертації

Основні положення роботи доповідались і обговорювались на :

1. XII Міжнародній науково-практичній конференції аспірантів, магістрантів, студентів «Сучасні проблеми наукового забезпечення енергетики» (м. Київ, 22-25 квітня 2018 року).

Публікації. Наукові положення дипломної роботи опубліковані у 1 роботі.

Ключові слова. *БАЗА ПРОГРАМНОГО КОДУ, СИСТЕМА КОНТРОЮ ВЕРСІЙ, MICROSOFT OFFICE 365, SHAREPOINT WEBPARTS*.

Список використаних джерел

1. XII Міжнародній науково-практичній конференції аспірантів, магістрантів, студентів «Сучасні проблеми наукового забезпечення енергетики» (м. Київ, 22-25 квітня 2018 року).

АНОТАЦІЯ

Магістерська робота присвячена розробці інструментів для інтеграції репозиторіїв системи керування версіями з внутрішніми даними системи у вигляді у вигляді компоненту до існуючої системи Microsoft Office 365. Програмний продукт являє собою компонент системи SharePoint WebParts створений у вигляді веб-застосунку з використанням мов програмування Javascript, HTML, CSS.

Система може використовуватися користувачами у сфері розробки програмних продуктів, що використовують можливості платформи Office 365. Потенційними користувачами програмної системи є керівники та виконавці завдань у сфері розробки складних інформаційних систем.

Ключові слова: база програмного коду, система контрою версій, microsoft office 365, sharepoint webparts.

ABSTRACT

The master's work is devoted to the development of tools for integrating repositories of the version control system with internal system data in the form of a component for the existing system of Microsoft Office 365. The software product is a component of the SharePoint WebParts system created as a web application using the programming languages Javascript, HTML, CSS.

The system can be used by users working in the software engineering field of products that use the capabilities of the Office 365 platform. Potential users of the software system are executives and executors of tasks in the field of development of complex information systems.

Keywords: code base, version control system, microsoft office 365, sharepoint webparts.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД	База даних
SQL	Structured query language — мова структурованих запитів
СУБД	Система керування базами даних
MVC	Model-view-controller — Модель–представлення–контролер
MVVM	Model-View-View-Model — Модель–представлення– представлення–контролер
HTML	Hypertext Markup Language — Мова розмітки гіпертекстових документів
DOM	Об'єктна модель документа
SPA	Single page application — Односторінковий застосунок
IDE	Integrated development environment — Інтегроване середовище розробки
VCS, СКВ	Version Control System — Система контролю версій
UI	User Interface — Інтерфейс користувача
ПК	Персональний комп'ютер
ОС	Операційна система
JSON	JavaScript Object Notation
AJAX	Asynchronous Javascript and XML

ВСТУП

Розробка складних інформаційних систем вимагає узгодженої роботи цілої групи програмістів. Проблеми організації взаємодії при розробці програмного продукту великою командою розробників зазвичай вирішуються використанням системи керування версіями програмного коду (Version Control System, VCS) та системами керування задачами [1].

В той же час система керування версіями не існує сама по собі, а тісно пов'язана із іншими внутрішніми інструментами та системами, що використовуються в процесі розробки та підтримки програмних продуктів. До них відносять системи документації, керування задачами (такими як Redmine, Jira), системами неперервної інтеграції (Jenkins) та внутрішніми продуктами компанії.

Наразі на кафедрі ведеться розробка компонентів для єдиної системи керування проектами на основі хмарного офісного пакету послуг Microsoft Office 365, що забезпечить користувачам доступ до документів і даних проектів через браузер з будь-якого пристрою з можливістю виходу в Інтернет. Проекти що передбачають розробку програмного продукту потребують також підтримки систем керування версіями програмного коду. Microsoft Office 365 не містить таких інструментів, але в той же час існують хмарні сервіси керування версіями програмного коду з вільним доступом, що можуть бути використані для спільного доступу до бази коду проекту. Але наразі в системі відсутня можливість інтеграції із віддаленими репозиторіями систем керування версій.

Для вирішення цієї проблеми пропонується розробити систему керування базою програмного коду у вигляді веб-застосунку (SharePoint WebParts).

Розроблена система дозволить пов'язати конкретні проекти в системі Microsoft SharePoint з репозиторієм коду та підвищити рівень і якість взаємодії розробників у команді, зменшити питомі витрати на синхронізацію цих змін з іншими, суміжними проектами.

1. СУЧАСНИЙ СТАН РОЗВИТКУ СИСТЕМ КОНТРОЛЮ ВЕРСІЙ ПРИ КЕРУВАННІ БАЗОЮ ПРОГРАМНОГО КОДУ

Система контролю дозволяє зберігати попередні версії файлів та завантажувати їх за потребою. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки.

Системи контролю версії надають ряд додаткових можливостей:

— можливість створення різних варіантів одного документу;
— документування всіх змін (коли ким було змінено/додано, хто який рядок змінив);
— реалізує функцію контролю доступу користувачів до файлів;
— дозволяє створювати документацію проекту з поетапним записом змін в залежності від версії;
— дозволяє додавати пояснення до змін та документувати їх.

Використання системи контролю версії є необхідним для роботи над великими проектами, над якими одночасно працює велика кількість розробників.

Місце зберігання даних файлів називають репозиторієм. В середині кожного з репозиторіїв можуть бути створені паралельні лінії розробки — гілки [2].

Гілки зазвичай використовують для зберігання експериментальних, незавершених(alpha, beta) та повністю робочих версій проекту(final).

Більшість систем контролю версії дозволяють кожному з об'єктів присвоювати теги, за допомогою яких можна формувати нові гілки та репозиторії.

Мета дослідження полягає в розробці автоматизованої системи керування базою програмного коду.

Для досягнення поставленої задачі були сформульовані наступні завдання дослідження, що визначили логіку дослідження та його структуру:

— проаналізувати існуючі методи контролю виконання задач;

— дослідити можливості аналогічних продуктів;

— проаналізувати можливості систем керування версіями програмного коду;

— пов'язати користувачів систем контролю версій з користувачами SharePoint на основі пари логін в SharePoint – логін в системах GitHub, Bitbucket, Gitlab;

— розробити програмний продукт у вигляді компоненту до існуючої системи.

На рисунку 1.1. відображена UML-діаграма прецедентів, що зображає функціональні можливості користувачів програмного продукту. Вона містить відношення між акторами (категорія адміністратора та звичайних користувачів системи) та прецедентами (функціональні можливості), що повинні бути реалізовані в компоненті.

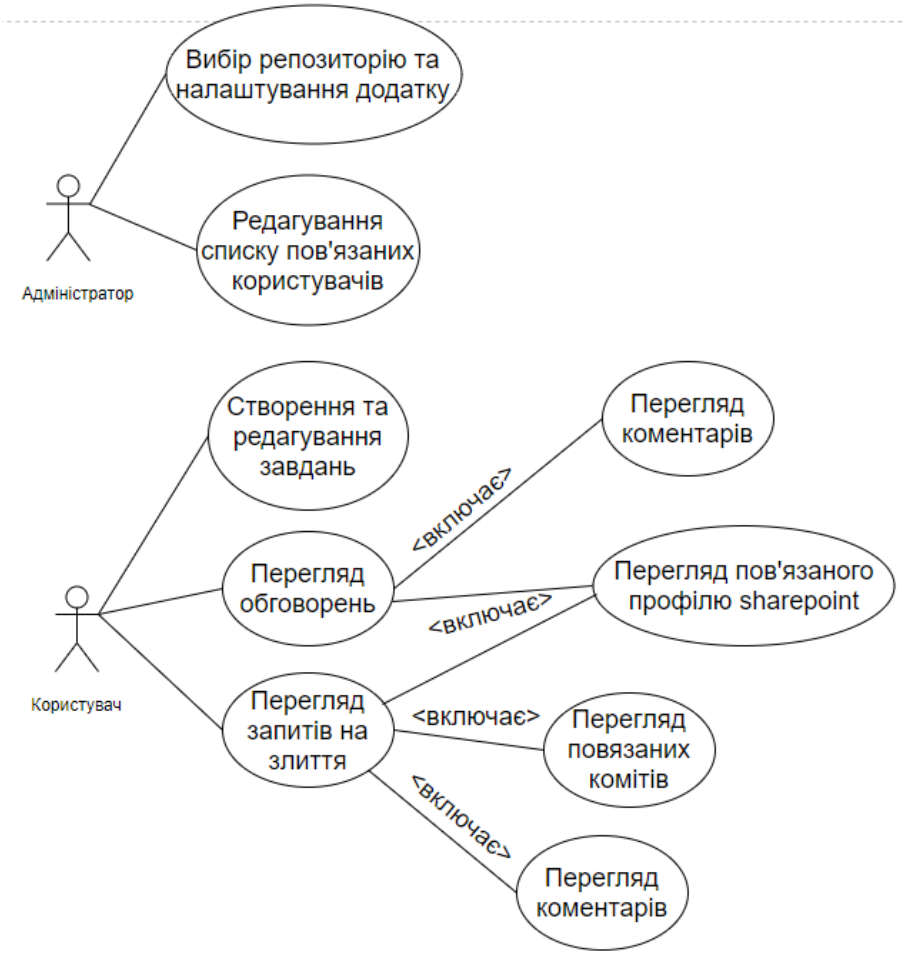


Рисунок 1.1— Функціональні можливості користувачів

Об’єктом дослідження є проблеми організації взаємодії при розробці програмного продукту великою командою розробників.

1.1 Системи контролю версій

Система контролю версій – це система, яка записує зміни в файл або набір файлів протягом часу і дозволяє повернутися пізніше до певної версії. Контроль версій файлів зазвичай використовується для вихідного коду програмного забезпечення, хоча насправді системи контролю версій підтримують будь-які типи файлів. Вона дозволяє повернути файли до стану, в якому вони були до змін, повернути весь проект до початкового стану, побачити всі внесені зміни за вказаний проміжок часу, побачити, хто останній змінював щось і тим самим викликав проблему, хто поставив завдання та коли, та багато іншого. Використання СКВ також означає, що, якщо після редагуванні вихідного коду програма перестала працювати правильно або якісь файли було втрачено в результаті комп'ютерного збою, ви спокійно можете все виправити. [3]

Основні можливості, що надають системи контролю версій включають:

— перегляд стану проекту в більш ранні моменти часу;
— показ відмінностей між різними станами та версіями проекту;
— розділення розвитку проекту на кілька незалежних ліній, званих "гілками", які можуть розвиватися незалежно один від одного;
— узгодження змін, внесених різними розробниками до різних гілок, так званим процесом "злиття";
— дозволяють багатьом людям працювати над одним проектом одночасно розділяючи і об'єднуючи результати їх роботи по мірі необхідності.

1.1.1 Локальні системи контролю версій

Багато людей в якості одного з методів контролю версій застосовують копіювання файлів в окрему директорію. Даний підхід є дуже поширеним завдяки

його простоті, проте він, неймовірним чином, схильний до появи помилок. Можна легко забути в якій директорії ви знаходитеся і випадково змінити не той файл або скопіювати не ті файли, які ви хотіли [1].

Щоб вирішити цю проблемою, спочатку було розроблено локальні СКВ, що представляють собою базу даних, яка зберігає версії файлів у вигляді змін, що внесені в файли (рисунок 1.2.).

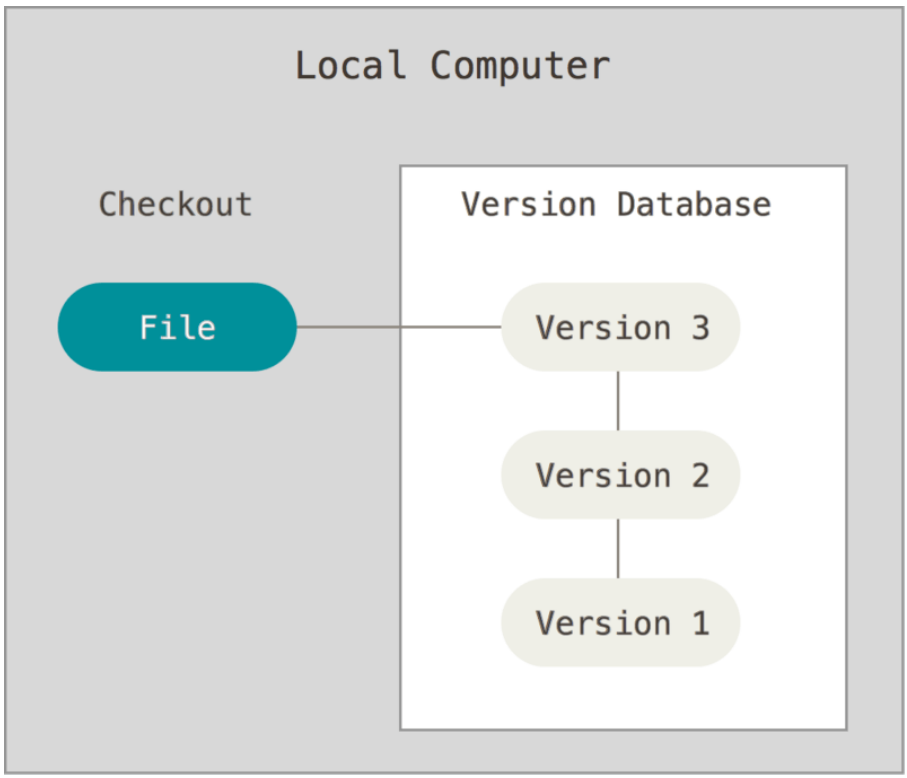


Рисунок 1.2 – Локальні системи контролю версій

Прикладом локальної СКВ є система під назвою RCS. RCS зберігає набори латок (тобто відмінностей між файлами) в спеціальному форматі на диску, шляхом додавання всіх латок можливо відтворити стару версію будь-якого файлу.

1.1.2 Централізовані системи контролю версій

Наступним важливим питанням, з яким стикаються люди, є необхідність співпрацювати з іншими розробниками. Щоб справитися з цією проблемою, були розроблені централізовані системи контролю версій (ЦСКВ).

Системи контролю версій, такі як CVS та Subversion є централізованими, що означає наявність єдиної центральної копії із вмістом та історією проекту, на яку повинні посилатися всі користувачі. Як правило доступ до неї отримують по мережі, і якщо центральна копія недоступна з якої-небудь причини, всі користувачі втрачають можливість використовувати керування версіями, поки центральна копія не запрацює знову. Такі системи як CVS, Subversion і Perforce, мають єдиний сервер, який містить всі версії файлів, та деяке число клієнтів, які отримують файли з центрального місця (рисунок 1.3.). Протягом багатьох років, це було стандартом для систем контролю версій.

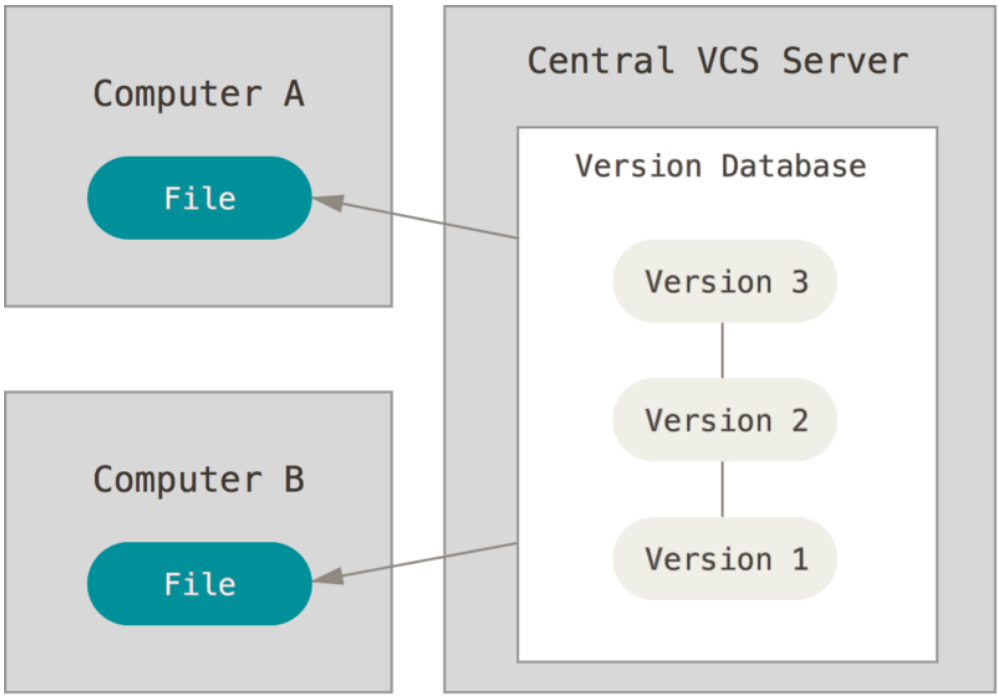


Рисунок 1.3 – Централізовані системи контролю версій.

Такий підхід має безліч переваг, особливо над локальними СКВ. Наприклад, кожному учаснику проекту відомо, певною мірою, чим займаються інші. Адміністратори мають повний контроль над тим, хто і що може робити. Набагато легше адмініструвати ЦСКВ, ніж мати справу з локальними базами даних для кожного клієнта.

Але цей підхід також має деякі серйозні недоліки. Найбільш очевидним є єдина точка відмови, якою є централізований сервер. Якщо сервер виходить з ладу

протягом години, то протягом цієї години ніхто не може співпрацювати або зберігати зміни над якими вони працюють у системі контролю версій. [4] Якщо жорсткий диск центральної бази даних на сервері пошкоджено, і резервні копії не були зроблені своєчасно, ви втрачаєте абсолютно все — всю історію проекту, крім одиночних знімків проекту, що збереглися на локальних машинах людей. Локальні СКВ теж мають цей недолік — щоразу, коли вся історія проекту зберігається в одному місці, є ризик втратити все.

1.1.3 Децентралізовані системи контролю версій

В ДСКВ (таких як, Git, Mercurial, Bazaar або Darcs), клієнти не просто отримують останній знімок файлів репозиторію: натомість вони є повною копією сховища разом з усією його історією (рисунок 1.4.).

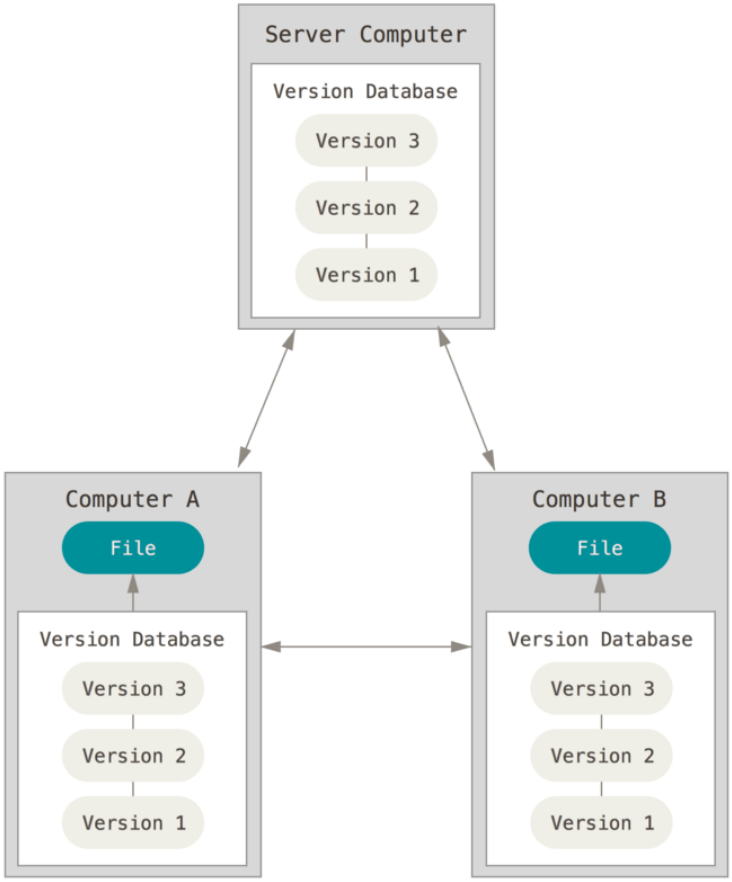


Рисунок 1.4 – Децентралізовані системи контролю версій.

Розподілені системи, такі як Git, з іншого боку, не мають власної центральної копії. Кожен користувач має повну, незалежну копію всієї історії проекту, так зване

"сховище", і повний доступ до всіх можливостей керування версіями. Доступ до мережі необхідний лише зрідка, для того щоб ділитися наборами змін з людьми, що працюють над цим проектом [5].

Таким чином, якщо сервер, через який співпрацюють розробники, перестає працювати, будь-яка клієнтська копія репозиторію може стати основою для нового серверу. [6] Кожна клієнтська копія насправді є повною резервною копією всіх даних.

Більш того, багато з цих систем дуже добре взаємодіють з декількома віддаленими репозиторіями, так що ви можете співпрацювати з різними групами людей, застосовуючи різні підходи в межах одного проекту одночасно. Це дозволяє налаштувати декілька типів робочих процесів, таких як ієрархічні моделі, які неможливі в централізованих системах.

1.1.4. Порівняльна характеристика сучасних систем контролю версій

В цьому розділі наведено порівняльну таблицю характеристик з найбільш розповсюджених СКВ, що сумарно займають приблизно 98% ринку програмного забезпечення контролю версій [7]

Таблиця 1.1. Порівняльна характеристика сучасних систем контролю версій

	Git	Svn	Mercurial
Тип	Децентралізована	Централізована	Децентралізована
Підтримка тегів	Є	Часткова	Є
Можливість редагування історії змін репозиторію	Можливо	Можливо	Неможливо
Популярність	Висока	Середня	Середня
Складність вивчення	Висока	Низька	Низька
Швидкодія	Висока	Середня	Висока
Сервіси що надають хостинг репозиторіїв	Github, Gitlab, Bitbucket	Github	Bitbucket, Mozdev

Висновки до розділу 1

1. На основі аналізу різних типів систем контролю версій зрозуміло, що децентралізовані системи забезпечують найкращу підтримку керування базою програмного коду, оскільки дозволяють співпрацювати з різними групами людей, застосовуючи різні підходи в межах одного проекту одночасно, а також потребують доступу до мережі тільки для того щоб ділитися наборами змін з людьми.
2. На даний час Git є найпоширенішою серед децентралізованих СКВ, більше спеціалістів мають досвід роботи з нею, тому входження в проект нових спеціалістів відбувається швидко.
3. На основі порівняння сучасних систем хостингу репозиторіїв для СКВ Git було визначено, що Github, Gitlab, Bitbucket є найбільш затребуваними, оскільки надають широкий функціонал для підтримки та забезпечують вільний доступ для розподіленої розробки над базою програмного коду.
4. Обрані системи передбачають програмний інтерфейс для доступу до інформації про зміни програмного коду через інтерфейс користувача та прикладний інтерфейс розробника на основі REST-архітектури, тому дозволяють інтегрувати їх з іншими системами.

2. ШЛЯХИ ІНТЕГРАЦІЇ СИСТЕМИ КЕРУВАННЯ ПРОЕКТАМИ НА ПЛАТФОРМІ OFFICE 365 З СИСТЕМАМИ КОНТРОЛЮ ВЕРСІЙ

У цьому розділі проведено докладний аналіз застосування різних систем контролю до задачі узгодженої роботи цілої групи програмістів та визначено основні вимоги до розроблюваної системи. Також було обрано та обґрунтовано вибір архітектури односторінкових застосунків, засобів реалізації та середовища розробки.

2.1 Організація процесу розробки програмного продукту

З розвитком мережі Інтернет процес розробки програмного забезпечення все більше зміщується до віртуальної взаємодії між учасниками проектних команд. В IT-аутсорсингу це може стосуватися як однієї команди в різних офісах в одному місті, так і декілька команд в різних містах, або навіть країнах. І в цьому випадку досить гостро постає проблема організації комунікацій серед членів команди в процесі роботи над проектом. Ключовим елементом організації комунікацій в команді є визначення того, хто з ким буде взаємодіяти і кому яка інформація буде передаватися [8]. Особливостями віддалених комунікацій є обмеженість каналів комунікацій та зменшення міжособистісного спілкування, як між членами проектної команди, так і зі замовниками чи керівництвом. Територіальна віддаленість та різниця у часових зонах знижує можливість синхронної комунікації навіть за допомогою програмних засобів. Крім того, різниця в часових зонах зменшує кількість годин, які команда може знаходитися в контакті.

Також, для територіально розподілених команд характерні проблеми ініціації процесу комунікації, коли член команди не знає до кого звернутися з питанням або

вибраний канал комунікації не дозволяє отримати відповідь. Для вирішення цих проблем можна надати наступні рекомендації: сформувати список контактів учасників проекту, поширити організаційну діаграму команди проекту, сформувати розклад мітингів, виділити канали комунікацій, вказати інструменти здійснення комунікацій, надати рекомендації щодо здійснення зворотного зв’язку тощо.

Зростання складності управління та необхідність ведення контролю за усіма видами діяльності підприємства зумовлює необхідність впровадження та використання інформаційних технологій, які дозволяють прискорити процес створення та продажу програмних продуктів та послуг споживачам, здійснювати маркетингове планування, фінансово-облікову й господарську діяльність та ін. У зв'язку з цим процес автоматизації підприємств індустрії, що пов'язані зі сферою розробки або підтримки програмного забезпечення є актуальним для будь-якого підприємства. Активне впровадження сучасних інформаційних технологій у діяльність такого підприємства є необхідною умовою його успішної роботи, оскільки точність, надійність, оперативність і висока швидкість обробки та передачі інформації визначає ефективність управлінських рішень у цій сфері [9]. Також виникає потреба у чітко прописаних та регламентованих правилах користування електронною поштою та написання листів.

Використання електронної пошти може бути досить ефективним каналом комунікації, проте спеціалізовані системи, що також включають в себе внутрішні соціальні мережі, вікі-сайти, списки завдань, scrum та agile дошки та інші підсистеми краще дозволяють задокументувати процес роботи над проектом. При цьому значно спрощуються процес навчання, необхідний для нових співробітників та скорочується час необхідний для інтеграції їх у внутрішні процеси компанії або проекту.

Також використання єдиної системи дозволяє значно зменшити час потрібний на отримання зворотного зв'язку.

Так, наприклад, для організації працюючого зворотного зв’язку встановлюється правило відповіді на будь-яке повідомлення на протязі 2 годин. Крім того, всі питання, які можуть бути вирішені за допомогою цього каналу

комунікації мають задаватися одразу, не чекаючи на синхронізацію з іншою стороною.

Також мають бути регламентовані способи подання інформації, зокрема, розміри та формати файлів. Ці всі заходи призводять до економії часу на комунікацію, і відповідно до зменшення витрат на комунікацію.

2.2 Управління проектами в системі Office 365

Office 365 дозволяє створювати та редагувати файли Word, OneNote, PowerPoint і Excel у браузері а також працювати разом над одним файлом на декількох пристроях і переглядати зміни в міру їх появи. За допомогою Microsoft Planner робоча група може легко створювати плани, упорядковувати та призначати завдання, обмінюватися файлами, спілкуватися на робочі теми та стежити за перебігом виконання роботи. [10]

Office 365 також включає центр для командної роботи Microsoft Teams. Teams – це комплексне рішення для спілкування в чаті й організації онлайн-нарад. З його допомогою можна проводити аудіо- та відеоконференції, з функцію спільного перегляду екрана а також спілкуватися в чаті з людьми всередині організації й поза нею. [11]

Іншим компонентом Office 365 є SharePoint Online який надає можливості по створенню веб-сайту організації і внутрішніх соціальних мереж для спілкування та взаємодії співробітників [12]. Сайти, які створюються на платформі SharePoint, можна застосовувати як сховища інформації про розробки окремих лабораторій та документів пов'язаних із цими розробками, а також використовувати для виконання веб-застосунків, таких як вікі і блоги.

Усі ці продукти поєднуються між собою, ви можете наприклад прикріплювати документи та презентації або інші до окремих сайтів або списків задач Sharepoint, ділитися замітками OneNote і т.д., це робить Office 365 ідеальною системою для управління проектами.

2.3 Інтеграція системи керування версій з системою Office 365

SharePoint, що є одним зі стандартних компонентів Office 365 надає базові можливості роботи з документами, в тому числі файлами програмного коду але також не може бути використаний в якості системи контролю версій через те що:

— не дозволяє порівнювати між собою версії одного і того ж файлу;

— нема можливості створення окремих гілок, які використовуються (у системах керування версій) для відокремлення роботи по розробці нового функціоналу від релізної версії програмного продукту;

— не показує зміни в наборі файлів як одну зміну (використовується для тестування системи та відміни небажаних змін) ;

— не дозволяє одночасно працювати з різними версіями всього проекту різним розробникам;

У зв`язку з цим виникає проблема ведення взаємодії як у середині команди при постановці завдань та подальшому їх виконанні програмістами, так і проблема відсутності єдиної системи звітності про виконану роботу перед замовником.

Для вирішення цієї проблеми пропонується розробити систему керування базою програмного коду у вигляді веб-застосунку (SharePoint WebParts). Перевагою використання WebParts є можливість розробки системи без використання центрального серверу, із застосуванням технології односторінкових застосунків (single-page application).

Розроблена система дозволить пов'язати конкретні проекти в системі Microsoft SharePoint з репозиторієм коду та підвищити рівень і якість взаємодії розробників у команді, зменшити питомі витрати на синхронізацію цих змін з іншими, суміжними проектами.

Програмний продукт повинен забезпечувати наступні можливості:

— можливість роботи з віддаленим репозиторіями програмного коду;

— підтримка різних систем контролю версій (git, tfs, svn);

— використання прошарку що дозволить абстрагуватися від реалізації програмного інтерфейсу окремого провайдеру репозиторіїв систем керування версіями.

2.4 Формати зберігання та обміну даних

Формат обміну даними - це текстовий формат, який використовується для обміну даними між платформами між різними системами. Прикладами таких форматів є XML, JSON, CSV.

У розробленій системі, формат JSON було використано для роботи з програмними інтерфейсами Microsoft Sharepoint, та зовнішніми сервісами віддалених репозиторіїв систем керування версій, а також при використанні технології AJAX, для отримання даних клієнтом з серверу, що не потребують оновлення сторінки. Перевагою цього формату є її розповсюдженість та вбудована підтримка серіалізації та десеріалізації у більшості мов програмування.

2.4.1 Формат JSON

JSON розшифровується як JavaScript Object Notation. За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур (рисунок 2.1.) [13].

```
{
   "firstName": "Іван",
   "lastName": "Коваленко",
   "address": {
       "streetAddress": "вул. Грушевського 14, кв.101",
       "city": "Київ",
       "postalCode": 21000
   },
   "phoneNumbers": [
       "044 123-1234",
       "050 123-4567"
   ]
}
```

Рисунок 2.1 – Приклад структури JSON.

JSON-це текстовий формат, який повністю незалежний від мови, але використовує угоди, знайомі програмістам сімейства мов C, включаючи C, C++, C#, Java, JavaScript, Perl, Python і багато інших. Ці властивості роблять JSON ідеальною мовою обміну даними.

В JSON будується на двох типах структур:

— Колекція пар ім'я/значення. У різних мовах це реалізується як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив.

— Упорядкований список значень. У більшості мов це реалізовано як масив, вектор, список або послідовність.

Дані у форматі JSON (RFC 4627) являють собою:

— JavaScript-об'єкти { ... };

— масиви [..];

— або значення одного з типів:

- рядок,
- число,
- логічне значення true/false,
- null.

Об'єкт – невпорядкований набір пар ім'я/значення. Об'єкт починається з "{" закінчується "}" . За кожним ім'ям слідує: (двокрапка), а пари ім'я / значення поділяються комами (рисунок 2.2.) [14].

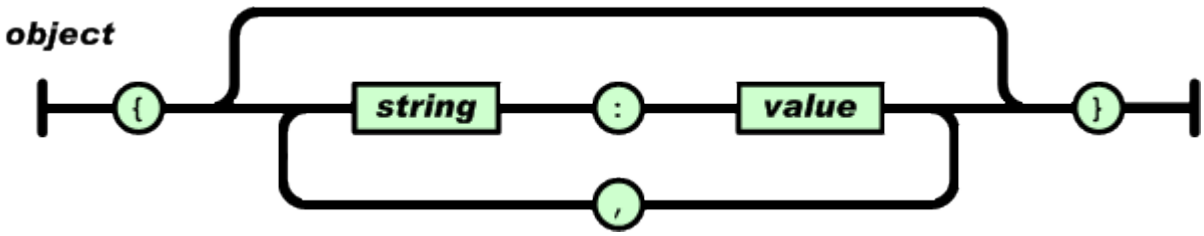


Рисунок 2.2 – Представлення об'єкту в JSON

Пари ключових значень мають двокрапки між собою, як наприклад тут "key" : "value". Кожна пара значень розділена комою, таким чином JSON, що містить 3 пари значень виглядає так: "key": "value", "key": "value"," key":"value" [15].

У більшості сценаріїв роботи з форматами обміну даними, дані створюються для відправки через Інтернет або мережу іншій стороні. Зазвичай для опису структури і типів даних використовується спеціальний файл JSON-схеми, на основі якого формується документація, що пояснює формат даних та надає приклади використання.

Перевагами використання JSON-схеми є:

— опис вже існуючої структури даних;
— можливість генерації документації на основі схеми;
— можливість перевірки даних, на основі схеми, що є корисним для автоматизованого тестування.

JSON-схема створюється у форматі JSON, приклад зображено на рисунку 2.3.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Cat",
  "properties": {
    "name": {
      "type": "string"
    },
    "age": {
      "type": "number",
      "description": "Your cat's age in years."
    },
    "declawed": {
      "type": "boolean"
    }
  }
}
```

Рисунок 2.3 – Структура JSON схеми.

Існує декілька спеціальних властивостей, що є обов'язковими при визначенні схеми:

Ключове слово "$schema" вказує, що ця схема написана відповідно до конкретного стандарту і використовується в першу чергу для управління версіями програмних інтерфейсів (API).

Ключове слово "$id" визначає URI для схеми і базовий URI, з яким дозволяються інші посилання URI в схемі.

Ключові слова анотації "title" та "description" носять тільки описовий характер. Вони не визначають обмежень до перевіряємих даних, однак корисні при формуванні документації.

Ключове слово "type" визначає обмеження типу даних що можуть міститися в документі JSON.

Властивість "properites" схеми повинно містити об'єкт зі списком полів, що описують структуру та тип даних JSON-документу. Кожне поле також може мати опис та бути необов'язковим, для цього в описі поля встановлюється властивість "required" зі значенням false.

2.4.2 Списки Sharepoint

Список — колекція даних в системі Sharepoint, якою можна ділитися з колегами та іншим користувачам сайту. SharePoint надає ряд готових до використання списків і шаблонів списків, які служать хорошою відправною точкою для організації елементів списків [16]. Загальний вигляд списку представлений на рисунку 2.4.

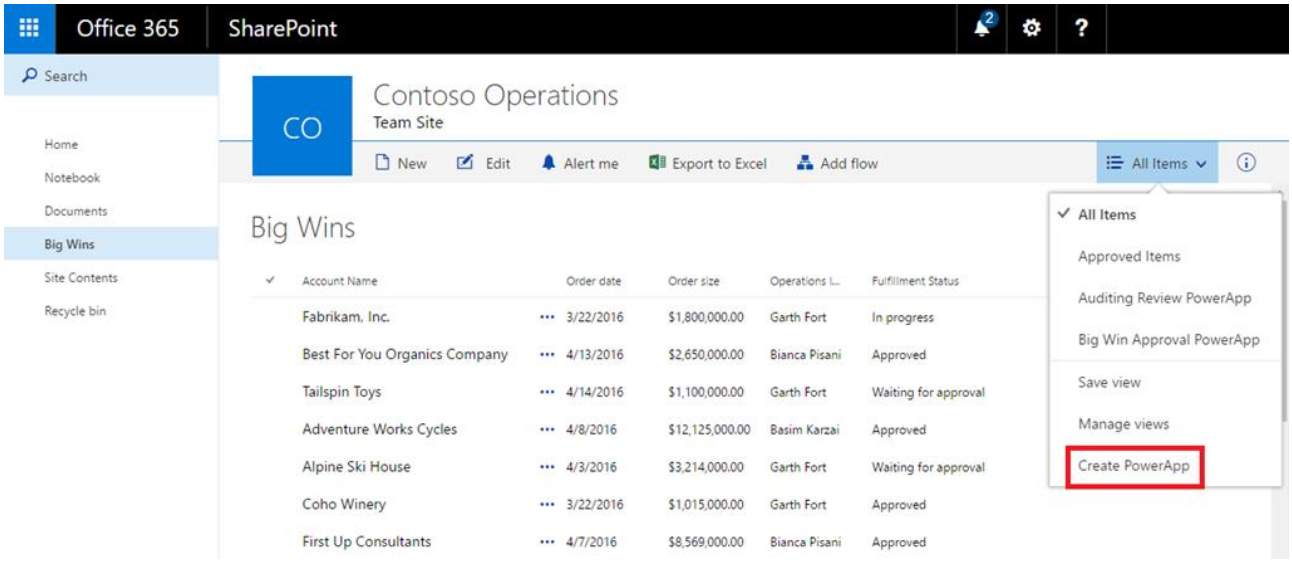


Рисунок 2.4 — Загальний вигляд списку

Списки — це потужний і гнучкий інструмент з великою кількістю вбудованих функцій, що представляє собою надійний спосіб роботи з даними, їх зберігання і спільного використання. [17]

Список може бути створений як із існуючого шаблону, що поставляється разом із інсталяцією Microsoft Sharepoint, або мати довільну структуру, визначену користувачем. По своїй сутності списки Sharepoint дуже схожі на таблиці традиційних баз даних, і мають поля певних типів, що визначаються при створенні або редагуванні структури списку. Поля можуть бути наступних типів:

— однорядковий текст;

— багаторядковий текст;

— вибір зі списку значень;

— числовий;

— грошове значення;

— дата і час;

— підстановка для вибору значень з іншого списку;

— булевий тип;

— посилання на користувача або групу;

— гіперпосилання або малюнок;

— обчислюваний, що формується на основі інших полів;

— статус завдання;

— посилання на зовнішні дані;

— керовані метадані.

Шаблони визначають не тільки структуру даних самого списку, але зазвичай також змінюють зовнішній вигляд форми для редагування додавання та перегляду даних (рисунок 2.5.) [18].

Нижче перераховані дії, які можна виконувати зі списками:

— до елементу списку можна прикріпити один або кілька файлів, щоб надати додаткові відомості, наприклад, електронну таблицю з номерами телефонів служби підтримки або документ з довідковою інформацією;

— можна створювати подання списку для впорядкування, сортування і фільтрації даних різними способами, змінювати метадані (наприклад, видаляючи і додаючи стовпці) і редагувати правила перевірки. Крім того, дані зі списків можна використовувати на внутрішніх сайтах. Наприклад, на домашній сторінці можна відобразити тільки поточні події з календаря, а на іншій-створити візуальне уявлення у вигляді настінного календаря;

— між списками можна створювати відносини, використовуючи поєднання унікальних стовпців-ідентифікаторів, випадаючих списків, для вибору значень з іншого списку, забезпечувати дії при видаленні одного з елементів зв'язаного списку (каскадне і обмежене видалення);

— можна створювати настроюванні списки, відображати дані у веб-частинах та на сторінках веб-частин, імпортувати дані з інших програм, таких як Excel і Access, а також експортувати їх і встановлювати з ними зв'язок;

— можна відстежувати версії і вести докладний журнал змін, вимагати затвердження зміни даних, забезпечувати безпеку на рівні елементів і папок, виконувати вилучення і повернення файлів, а також автоматично отримувати повідомлення про зміни за допомогою повідомлень електронної пошти і RSS-каналів;

— можна упорядкувати контент одного списку по папках для зручності і підвищення продуктивності, а також підвищити загальну швидкодію шляхом індексації великих списків.

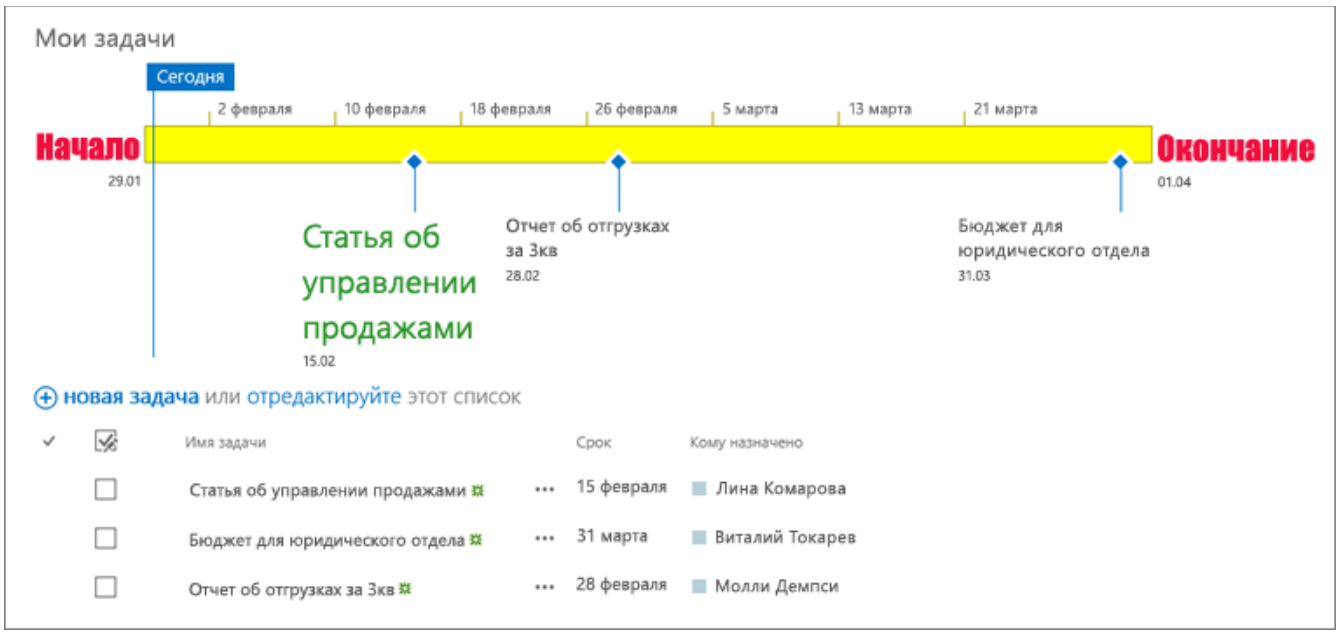


Рисунок 2.5 — Список завдань

Наразі існують наступні вбудовані шаблони списків типів [19]:

— сповіщення – використовується для повідомлення про новини та нагадування, підтримує розширені можливості форматування, включаючи малюнки, гіперпосилання і форматований текст;

— контакти – список контактів використовується для зберігання інформації про людей або груп, підтримує імпорт даних з Outlook;

— дошки обговорень – використовується для запису і централізованого зберігання обговорень робочої групи аналогічно груп новин, якщо адміністратор налаштував повідомлення електронної пошти списками, то в дошках обговорень можна зберігати обговорення, що ведуться по електронній пошті [20];

— посилання – використовується для зберігання посилань на ресурси з Інтернету, інтрамережі вашої компанії та інші ресурси;

— календар – надає візуальне представлення календару подій групи, включаючи збори, соціальні заходи, також можна відстежувати робочі процеси групи, наприклад дедлайни або дати випуску продуктів (рисунок 2.6.). Підтримує імпорт та експорт даних з Outlook;

— завдання – використовується для відстеження відомостей про проекти та інші події і цілі групи. Завдання можна призначати для користувачів, а також відстежувати стан і відсоток виконання. При використанні електронної пошти або програм для управління завданнями, що сумісні з технологіями SharePoint завдання можна переглядати та оновлювати з сайту SharePoint і вони автоматично будуть оновлені в цих програмах;

— завдання проекту – для зберігання відомостей про завдання з використанням діаграми Ганта, можуть відстежувати стан і відсоток завершення завдання;

— відстеження питань – для зберігання відомостей про певні питання, наприклад пов'язані зі службою підтримки, а також для відстеження ходу їх обробки. Можна призначати питання, розподіляти їх за категоріями і пов'язувати один з одним. При редагуванні питань дозволяється додавати примітки, ведучи таким чином історію приміток. Список відстеження питань також можливо використовувати в поєднанні з трьохетапним робочим процесом [21];

— опитування – при наявності електронної таблиці або бази даних, сумісної з технологіями SharePoint, дозволяється експортувати результати для подальшого аналізу їх;

— зовнішні списки – для роботи з даними, що зберігаються поза SharePoint, але їх можна читати і записувати їх в межах SharePoint;

— циркуляр – використовуються для розсилки учасникам робочої групи різних відомостей, включаючи штампи підтвердження.

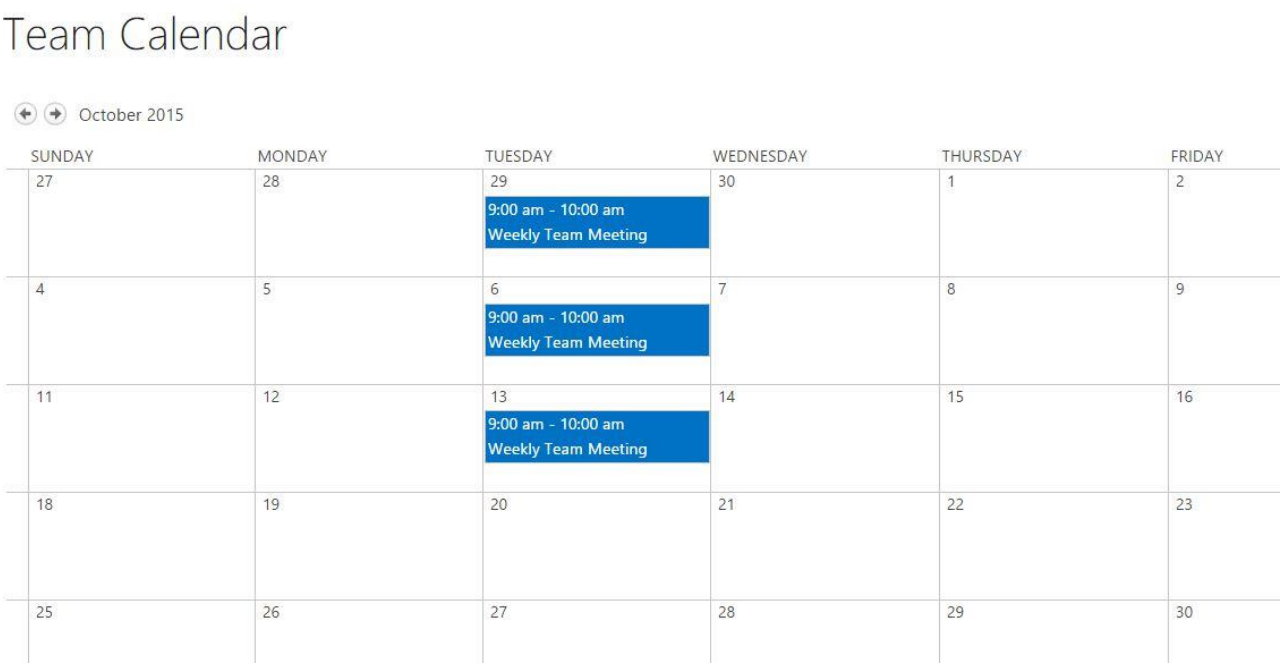


Рисунок 2.6 — Список типу календар

2.5 Особливості архітектури односторінкових застосунків

Для розробки компоненту системи для доступу до віддалених репозиторіїв бази програмного коду було обрано архітектуру односторінкових застосунків.

Односторінкові застосунки (SPA) – це програми, які завантажують одну сторінку HTML та динамічно оновлюють цю сторінку, на основі отриманих даних, коли користувач взаємодіє з додатком. Це також означає, що більша частина роботи відбувається на стороні клієнта, в JavaScript, сервер тільки надає данні, зазвичай у форматі JSON, а вся обробка та генерація сторінок виконується у браузері [5].

2.5.1 Фреймворки для розробки односторінкових застосунків

До початку 2000-х років, браузери не мали можливостей, які вони мають зараз. Вони були набагато менш потужними, і створення складних додатків всередині них не було можливим з точки зору продуктивності. Зі зростанням інтерактивності сторінок, що відображаються у браузері також зростала і складність

клієнтського програмного коду, частина бізнес-логіки була перенесена з серверної частини в клієнтську.

Такі бібліотеки, як jQuery і Mootools, були першими великими проектами, побудованими на JavaScript. Вони надавали більш приємний API для взаємодії з браузером і реалізували власні методи які дозволяли позбутися помилок і невідповідностей поведінки коду для різних браузерів. Такі фреймворки як Backbone, Ember, Knockout, AngularJS були першою хвилею сучасних JavaScript-фреймворків. До другої хвилі сучасних фреймворків відносять React, Angular і Vue в якості основних представників. Фреймворки абстрагують взаємодію з браузером і DOM. Замість того, щоб маніпулювати елементами, посилаючись на них в моделі DOM, ми декларативно визначаємо їх і взаємодіємо з ними на більш високому рівні, а сам фреймворк відповідає за їх відображення, це також дозволяє значно покращити продуктивність оновлення даних [22].

На сьогоднішній день існує безліч фреймворків, які націлені на досягнення найкращих результатів при створенні проектів різного розміру простим і приємним способом [23]. Одним з таких фреймворків є Vue.js, його превагою э те що він є досить простим, коли це потрібно та при цьому може запропонувати схожі можливості до інших фреймворків таких як React або Angular.

2.5.2 Механізм шаблонів

Для генерації результуючої сторінки в клієнтських фреймворках використовується механізм шаблонів, що дозволяє згенерувати HTML сторінку використовуючи заздалегідь визначений шаблон та дані, що можуть змінюватися (рисунок 2.7.). Кодогенерація активно використовується при генерації змісту на веб-сайтах, тому всі мови програмування, що використовуються для розробки веб-сайтів мають власні програмні реалізації механізму шаблонів, прикладами таких є PHP, Perl, Java, фреймворки ASP.NET та Ruby On Rails [24].

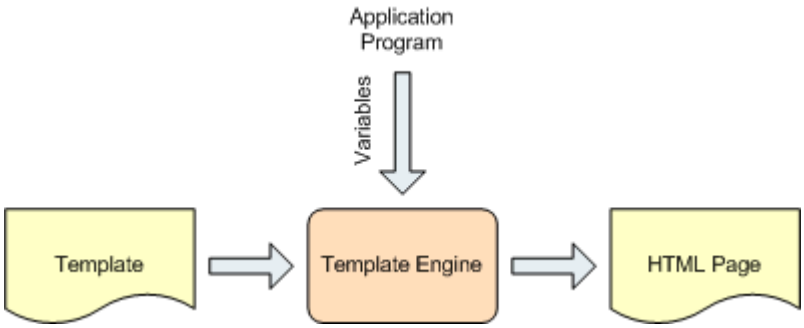


Рисунок 2.7 — Приклад роботи шаблонів

Наприклад фреймворк Vue.js використовує мову шаблонів, яка є надмножиною HTML. Це означає що будь-який коректний код HTML є допустимим в шаблонах, і надає дві потужні можливості: інтерполяцію і директиви [22].

Директиви — це спеціальні атрибути в мові розмітки документа з префіксом v-. В якості значення вони приймають один вираз JavaScript. Директива змінює об'єктну модель документу при оновленні значення цього виразу [25].

Інтерполяція дозволяє використовувати будь-який вираз JavaScript у шаблонах, для цього використовується синтаксис подвійних фігурних дужок — "{{expression}}". Краще всього уникати додавання складної логіки в шаблони, але той факт, що Vue дозволяє це, дає нам більше гнучкості, особливо при тестуванні. За допомогою інтерполяції можна також звертатися до змінних та викликати методи що визначені в екземплярі Vue.js.

2.5.3 Рендеринг на стороні серверу

На відміну від традиційних веб-сайтів, сервери яких повертають вже згенерований код HTML, зрозумілий для браузеру користувача, односторінкові застосунки виконують усю обробку даних в браузері, що означає що початковий запит сервера зазвичай повертає порожній HTML-файл з купою посилань CSS і JavaScript (JS). Потім зовнішні файли повинні бути завантажені та виконані для відображення відповідної розмітки. [26]

Рендеринг на стороні серверу є одним зі способів підвищення швидкості завантаження сторінок при першому візиті користувача. Користувачі отримують повну сторінку з видимим вмістом при завантаженні сайту, на відміну від порожньої сторінки, яка не заповнюється до запуску JavaScript.

Перевагами використання рендерингу на стороні серверу є [27]:

— краща індексація пошуковими системами, оскільки пошукові роботи бачать повністю сформовану сторінку, що відображає свій вміст навіть при вимкненому Javascript;
— підтримка застарілих браузерів та тонких клієнтів (наприклад пошукових роботів що здійснюють індексацію сторінок), що не підтримують виконання коду Javascript;
— зменшення часу, необхідного для відображення контенту (time-to-content), особливо при поганому з'єднанні із мережею інтернет або на повільних пристроях. Для розмітки, сформованої на сервері, не потрібно чекати поки весь JavaScript буде завантажений і виконаний, тому користувач побачить текст сторінки раніше. Як правило, це призводить до кращого користувацького досвіду і може бути критичним для додатків, де час до відображення контенту безпосередньо пов'язано з коефіцієнтом конверсії.

Слід враховувати і деякі компроміси при використанні серверного рендерингу:

— обмеження при розробці – код тільки для браузера може бути використаний лише на певних стадіях життєвого циклу додатку, деякі зовнішні бібліотеки можуть потребувати додаткової доробки для того, щоб мати можливість запускатися в додатку з серверним рендерингом;
— ускладнення процесу налаштування та розгортання збірки – на відміну від повністю статичного SPA, який може бути встановлений на будь-якому статичному файловому сервері, програми з серверним рендерингом вимагає оточення, де є можливість запустити сервер Node.js (рисунок 2.8.);

— підвищене навантаження на стороні сервера. Рендеринг повноцінного додатка в Node.js очевидно більш вимогливий до ресурсів процесора, ніж проста роздача статичних файлів, тому якщо необхідно обслуговувати велику кількість клієнтів одночасно, треба бути готовим до відповідного навантаження на сервер і використовувати стратегії кешування.

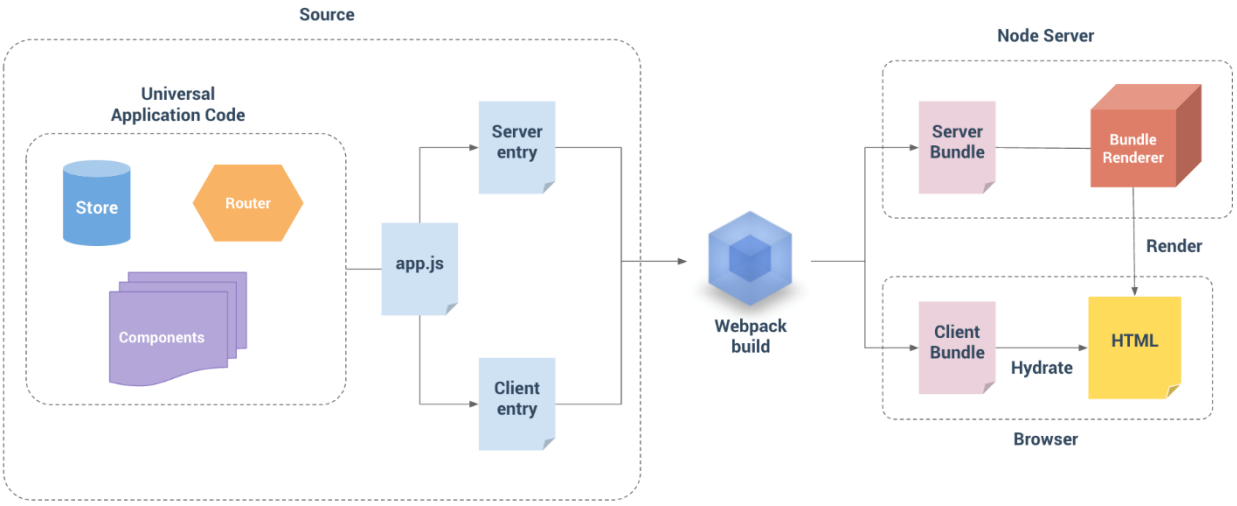


Рисунок 2.8 — Архітектура процесу збірки

2.5.4 Фреймворк Vue.js

Усі елементи користувацького інтерфейсу було оформлено у вигляді окремих компонентів фреймворку Vue.js, що дозволило значно знизити складність коду.

Vue називається прогресивним фреймворком. Це означає, що він адаптується до потреб розробника. У той час як інші фреймворки вимагають повної інтеграції розробників або команди і часто потребують, переписування існуючого коду з нуля, так як вони вимагають обов`язкового дотримання певного набору конвенцій розробки, Vue.js дозволяє використовувати себе лише в певних місцях програми, за допомогою використання простого тегу script, для початку розробки. Розробникам не потрібно розбиратися та налаштовувати webpack, Babel, npm або про інші, обов'язкові для React або Angular інструменти. Це одна з переваг Vue.js, особливо в нинішній екосистемі фреймворків та бібліотек JavaScript, яка має тенденцію

надмірно ускладнювати процес для новачків а також досвідчених розробників, що раніше працювали з іншими технологіями.

Vue був створений Эваном Ю (Evan You), коли він працював в Google над AngularJS (Angular 1.0) через необхідність створення більш швидких додатків. Vue успадкувати кутовий синтаксис шаблонів, але видалив складний стек технологій, який вимагав Angular, і робив його дуже ефективним. Одна з головних відмінностей від фреймворку React – відмова від використання JSX. Хоча технічно можна використовувати JSX в Vue, це не є популярним підходом, і замість цього використовується система шаблонів. Будь-який HTML-файл є дійсним шаблоном Vue, тоді як JSX сильно відрізняється від HTML і потребує додаткового навчання для людей в команді, які раніше працювали тільки з HTML-частиною додатку, наприклад дизайнери. Шаблони Vue багато в чому схожі на мови шаблонів Mustache і Handlebars (хоча і відрізняються гнучкістю) і тому вони більш знайомі розробникам, які вже використовували такі фреймворки, як Angular і Ember.

Ключовою особливістю дизайну Vue є його реактивність. При зміні даних згенерований шаблон HTML автоматично оновлюється відповідно до цієї зміни. Моделі даних являють собою прості JavaScript-об'єкти. У міру їх зміни оновлюється і представлення даних (рисунок 2.9.), завдяки чому управління станом програми стає простим і очевидним.

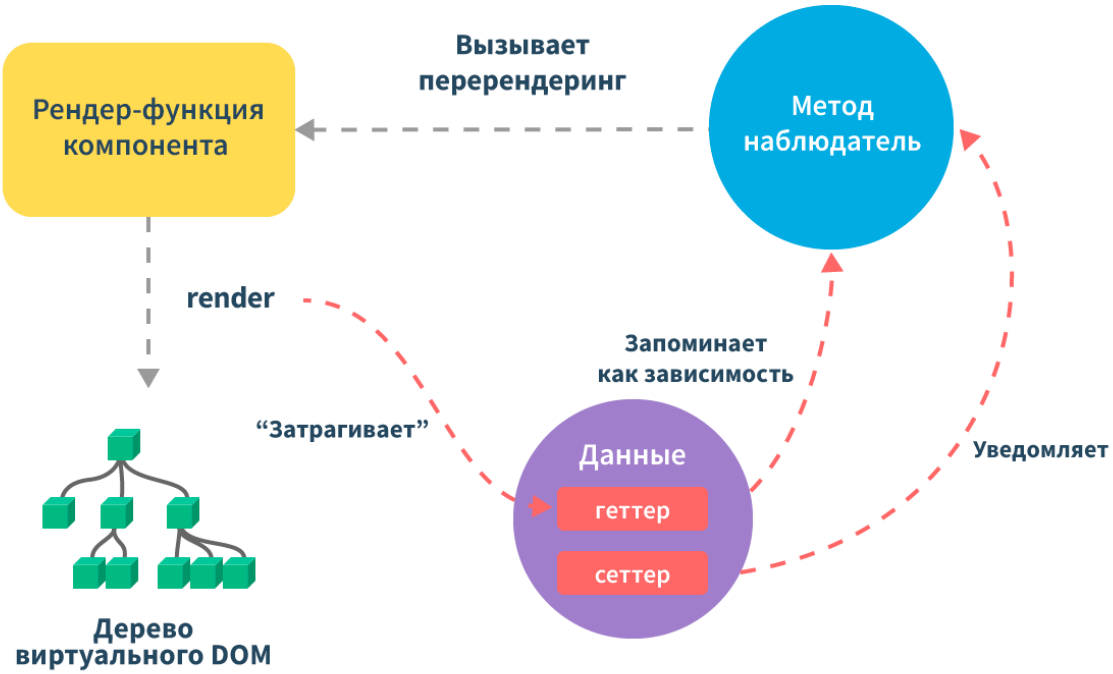


Рисунок 2.9 — Схема відстежування змін

Як і в інших фреймворках для розробки односторінкових застосунків в Vue існує поняття компонентів. Компоненти є єдиними, незалежними одиницями інтерфейсу. Вони можуть мати власний стан, розмітку і стиль. Компоненти - це повторювані екземпляри Vue зі своїм ім'ям. Компоненти дозволяють розробляти базові елементи інтерфейсу, що потім можна використовувати в різних місцях програми, працюючи з ними так само як і з стандартними елементами мови HTML: списками, кнопками, зображеннями, вспливаючими вікнами і т.д. Компоненти також можуть створювати власні типи подій, на які можна реєструвати власні обробники подій. На рисунку 2.10. зображено код та приклад компоненту – кнопки що рахує кількість натискань.

```
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">Счётчик кликов — {{ count }}</button>'
})
```

HTML

```
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

JS

```
new Vue({ el: '#components-demo' })
```

Счётчик кликов — 3

Рисунок 2.10 — Приклад компоненту

Зазвичай додаток організовується у вигляді дерева вкладених компонентів (рисунок 2.11).

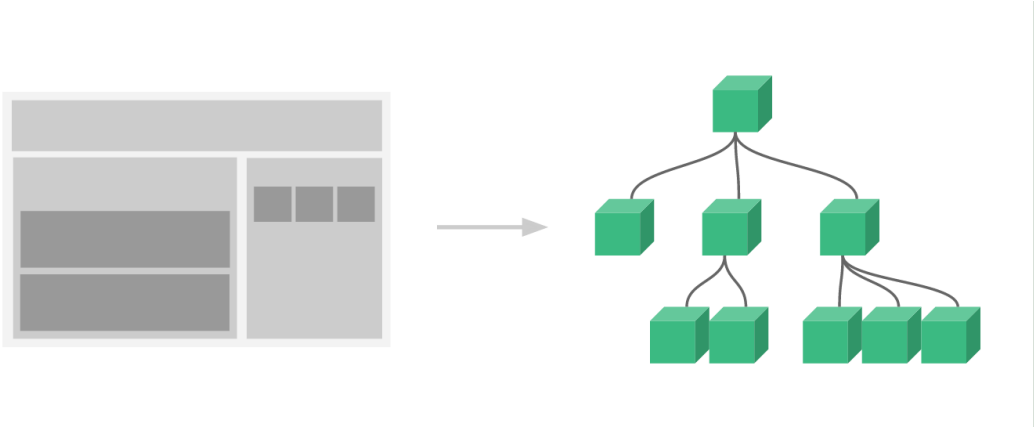


Рисунок 2.11 — Організація компонентів

Наприклад, у вас можуть бути компоненти для заголовка, бічній панелі, зони контенту, кожен з яких може містити інші компоненти для навігаційних посилань,

постів блогу і т. д. Є два типи реєстрації компонентів: глобальний і локальний. Компоненти, зареєстровані глобально, можуть використовуватися в шаблоні будь-якого кореневого екземпляру Vue (new Vue) — і навіть всередині всіх компонентів, розташованих в дереві компонентів цього примірника Vue. Локальні компоненти треба ж спочатку явно імпортувати, це дозволяє мати декілька компонентів з однаковою назвою, які не будуть конфліктувати між собою.

У багатьох проектах, глобальні компоненти визначаються за допомогою Vue.component, з подальшим new Vue({ el: '#container' }) для вказівки елементу-контейнера в тілі кожної сторінки. Для проектів малого та середнього розміру, в яких JavaScript використовується лише для деяких сторінок, цей підхід може прекрасно працювати. У більш складних проектах або у випадках, коли весь фронтенд управляється JavaScript, явними стають такі недоліки:

— глобальне визначення змушує давати унікальне ім'я кожному компоненту;
— рядковим шаблонам не вистачає підсвічування синтаксису. крім того, доводиться використовувати символ "\" для екранування багаторядкового html;
— немає модульної підтримки CSS — в той час як html і javascript розбиваються на модулі-компоненти;
— відсутність кроку збірки обмежує нас тільки html і es5 javascript, не дозволяючи використовувати препроцесори на кшталт pug і babel.

Існують способи запису визначень шаблонів в HTML-файл, але в цьому випадку виникає складність розділення розмітки та логіки [28]. Для вирішення цієї проблеми пропонується використовувати однофайлові компоненти. Вони мають розширення ".vue" і інкапсулюють шаблон компонента, код JavaScript і стилі CSS в одному файлі (рисунок 2.12.).

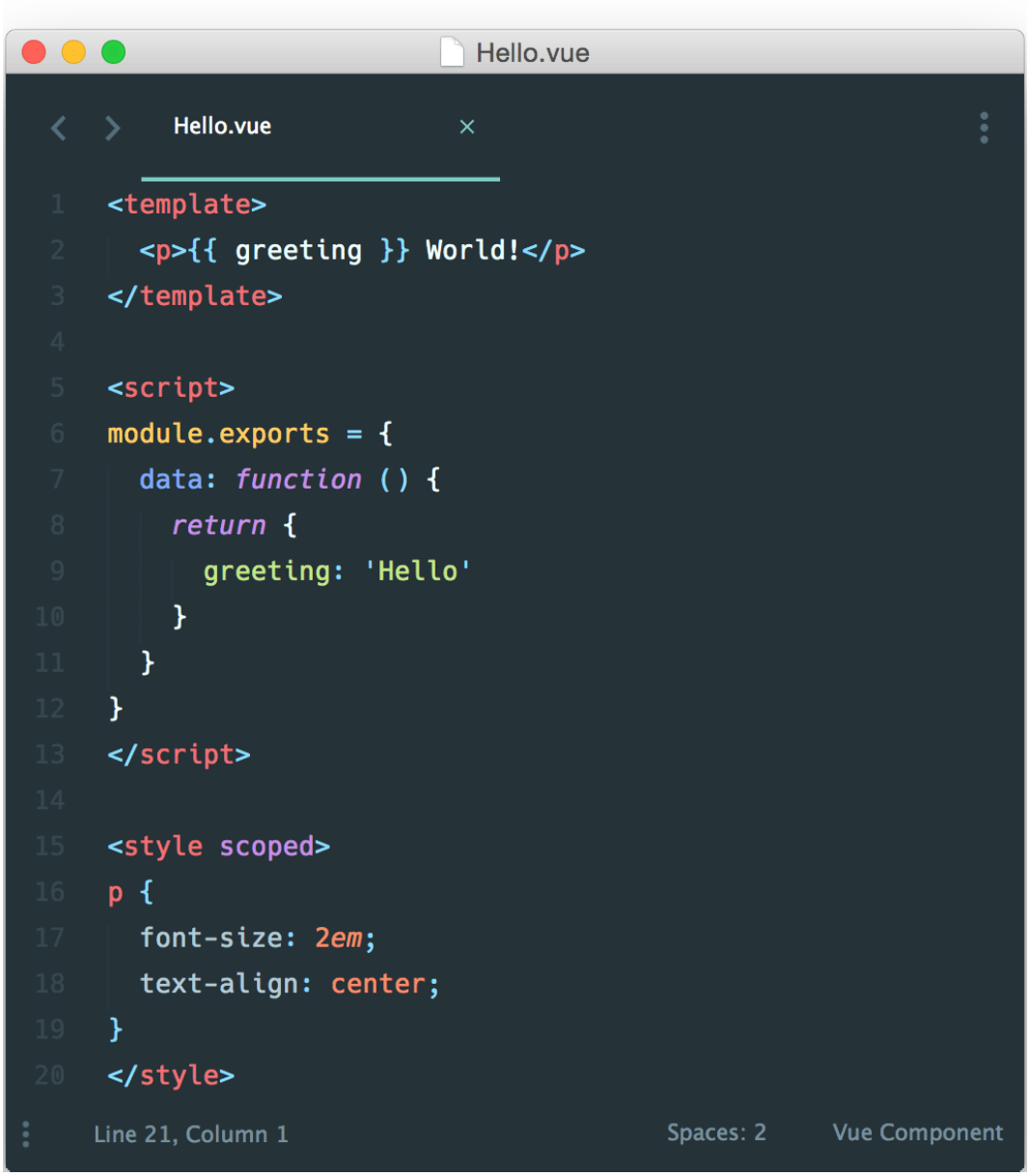


Рисунок 2.12 — Приклад однофайлового компоненту

Необхідно відзначити, шо поділ відповідальності це не те ж саме, що поділ на файли за типом. У сучасній розробці UI замість поділу кодової бази на три шари, що тісно переплітаються один з одним, має більше сенсу ділити їх на слабко пов'язані компоненти і компонувати вже їх. Всередині компоненту, його шаблон, логіка і стилі нерозривно пов'язані між собою, що дозволяє зробити компонент більш зв'язаним і спростити його підтримку.

Коли Vue.js використовується для формування HTML-сторінок, існує можливість реагувати на певні події користувача.

В Vue події можуть представляти не тільки базові події документу але й визначені користувачем подій, що відбуваються в окремих компонентах. Для визначення обробнику подій DOM використовується директива v-on. [29]

Vue.js також надає можливості рендерингу на стороні сервера за допомогою використання бібліотеки Nuxt.js [30].

2.6 Інтегроване середовище розробки ПЗ WebStorm 2017

Основним середовищем розробки було обрано інтегроване середовище розробки програмного забезпечення JetBrains WebStorm.

JetBrains WebStorm — інтегроване середовище розробки на JavaScript, CSS & HTML від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA (рисунок 2.13.) [31].

Рисунок 2.13 — Середовище розробки JetBrains WebStorm

Основні можливості:

— інтелектуальний редактор з підтримкою синтаксису, пошуку в документації та рефакторингу для мов та середовищ: JavaScript, Node.js, ECMAScript 6, TypeScript, CoffeeScript і Dart, а також для HTML, CSS, Less, Sass і Stylus;
— аналіз коду "на льоту", виділення помилок і швидкі виправлення;
— потужна навігація по проекту і розширений рефакторинг;
— підтримка сучасних фреймворків: React, Angular, AngularJS, Vue.js, Express, і багато іншого;
— вбудований відладчик для клієнтського коду і Node.js;
— інтеграція з інструментами збірки (Grunt, Gulp), якості коду (JSHint, JSLint, ESLint, TSLint), тест раннерів (Karma, Mocha, Jest, Protractor), і систем контролю версій (Git, Github, Mercurial, SVN).

WebStorm надає розширену підтримку фреймворків Angular, React, Vue.js і Meteor, React Native, PhoneGap а також Cordova і Ionic для мобільної розробки і розробки на стороні сервера з Node.js.

WebStorm аналізує проект, щоб забезпечити найкращі результати автодоповнення коду для всіх методів, функцій, модулів, змінних і класів, визначених у додатку. Допомога в кодуванні залежить від контексту і може також залежати від конкретної структури.

Всі помилки і попередження виводяться прямо в редакторі при введенні, та надають опції для швидкого виправлення цих помилок. Можна також запустити аналіз якості коду для всього проекту і автоматично застосувати вибрані швидкі виправлення.

IDE дозволяє налаштувати стиль коду для будь-якої мови, включаючи відступи, правила вирівнювання, переноси строк і т. д. Власний стиль проекту можна зберігати у файлі, та ділитися з іншими розробниками, що працюють над проектом через системи контролю версій.

WebStorm надає розширений відладчик для клієнтського коду, який працює разом з Google Chrome. Він вбудований прямо в IDE, тому вам не потрібно перемикатися між редактором і браузером для налагодження.

Можливо відлагоджувати код ECMAScript 6, TypeScript або CoffeeScript, покладаючись на підтримку відладчика WebStorm для вихідних карт.

Відладчик WebStorm має кілька областей видимості, включаючи фрейми, глобальні та локальні змінні. Значення змінних відображаються в рядку поруч з їх використанням в редакторі. Це дозволяє переглядати значення складних виразів JavaScript під час виконання. Точки зупинки підтримують режим призупинення та умов для зупинки.

WebStorm може скомпілювати код TypeScript в JavaScript за допомогою вбудованого компілятора. Параметри компіляції можна вказати вручну або в tsconfig.JSON-файлі. Всі помилки компіляції відображаються в редакторі на льоту.

WebStorm надає уніфікований інтерфейс для роботи з багатьма популярними системами керування версіями, забезпечуючи узгоджений користувацький інтерфейс в git, SVN, Mercurial і Perforce.

Всі незафіксовані зміни з системи контролю версій підсвічуються в лівій області редактора і у вікні проекту. Існує можливість легкої відміни будь-яку зміни всього за два кліка. Вбудований візуальний інструмент злиття швидко і інтуїтивно вирішує всі конфлікти.

2.7 Мова програмування TypeScript

Так як кожен віддалений провайдер репозиторіїв надає власний програмний інтерфейс доступу, було розроблено єдиний клас-інтерфейс для отримання даних та класи що реалізують окремі структури даних засобами інтерфейсів мови Typescript.

TypeScript — мова програмування, представлена Microsoft восени 2012, що базується на основі JavaScript. Популярність і актуальність ідей нової мови призвела до того, що ряд з цих ідей в подальшому стануть частиною нового стандарту

JavaScript. А нова версія одного з найпопулярніших фреймворків для Web - Angular 2/4/5/6 повністю написана на TypeScript спільно компаніями Microsoft і Google [32].

Розробником мови є Андерс Гейлсберг також відомий як творець таких мов як Delphi, C# та Turbo Pascal [33].

TypeScript забезпечує безпеку типів під час компіляції для коду JavaScript. Так як мова є зворотньо сумісною з JavaScript типи є повністю необов'язковими. Фактично, після компіляції програми код можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js. TypeScript навмисно і строго є надмножиною JavaScript з додатковою перевіркою типів. Це також дозволяє використовувати усі бібліотеки написані на JavaScript. Більш того, можна залишити існуючі JavaScript-проекти в незмінному вигляді, а дані про типізації розмістити у вигляді анотацій в окремих файлах, які не заважатимуть розробці і прямому використанню проекту [34].

Однак це не означає, що для написання коду вам не потрібно знати JavaScript. TypeScript просто стандартизує всі способи надання хорошої документації по JavaScript.

На відміну від інших мовами, що компілюються в Javascript, Typescript:

— не надає новий синтаксис мови для існуючих можливостей Javascript, адже це не допомагає виправити помилки (прикладом є мова CoffeeScript);
— не є мовою, що кардинально відрізняється від Javascript, тобто не абстрагує занадто далеко від середовищ виконання та спільнот розробників (прикладом є мова програмування Dart).

TypeScript надає ряд функцій, які плануються в наступному стандарті ES6 мови JavaScript (рисунок 2.14.), навіть для середовищ, які не підтримують ці функції для коду Javascript (старі веб-браузери та середовища виконання, такі як Node.js).

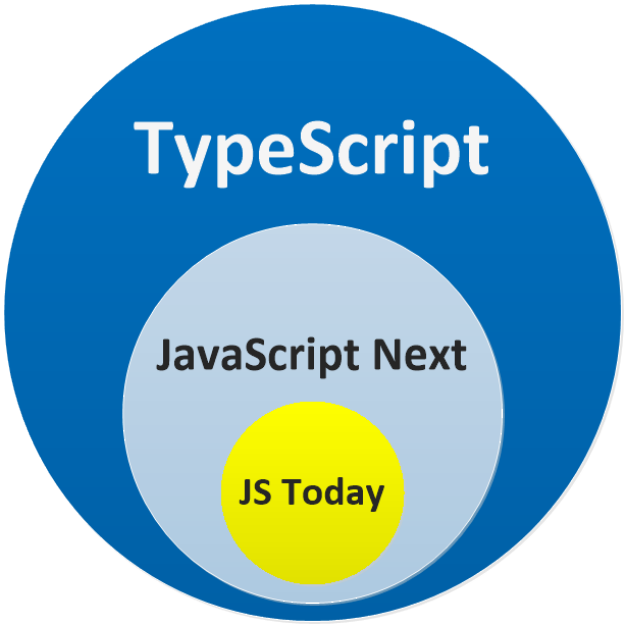


Рисунок 2.14 — Javascript є підмножиною мови Typescript

Оскільки дана мова є Open Source, то всі її інструменти доступні для всіх бажаючих. Для роботи з TypeScript можна використовувати як Windows, так і Linux і MacOS. Для написання коду на цій мові можна використовувати будь-який, навіть найпростіший текстовий редактор. Багато текстових редакторів і середовищ розробки, наприклад, Code Visual Studio, Atom, Sublime, Visual Studio, Netbeans, WebStorm та інші, мають підтримку TypeScript на рівні плагінів, що дозволяє скористатися низкою переваг, наприклад, підсвічуванням синтаксису або спливаючими підказками для типів і конструкцій мови. [35].

Типи можуть бути визначені неявними - компілятор спробує вивести якомога більше інформації про тип, щоб забезпечити безпеку типів з мінімальними витратами на продуктивність при розробці коду. Наприклад, у наступному прикладі TypeScript буде знати, що foo має номер типу число, і видасть помилку при спробі присвоїти змінній значення типу рядок (рисунок 2.15.).

```
var foo = 123;
foo = '456'; // Error: cannot assign `string` to `number`

// Is foo a number or a string?
```

Рисунок 2.15 — Приклад неявної типізації

Типи також можуть бути визначено явно, за допомогою анотації типів.

Анотації можна використовувати для:

— допомоги з визначення типом компілятору, при використанні складних конструкцій і, що більш важливо, для документації коду для наступного розробника, який буде працювати з цим кодом;
— примусового визначення типу змінної, через те, що при використанні неявних типів ваше розуміння коду не завжди відповідає алгоритмічному аналізу коду зробленому компілятором (інколи існує потреба в тому щоб змінна могла набувати значення одного з декількох типів, наприклад числового або булевого значення).

TypeScript використовує постфіксні анотації типів (рисунок 2.16.), популярні в інших опціонально анотованих мовах (наприклад, ActionScript і F#).

```
var foo: number = 123;

var foo: number = '123'; // Error: cannot assign a `string` to a `number`
```

Рисунок 2.16 — Приклад явної типізації

Через те що TypeScript позиціонується як мова, що є дуже схожою на JavaScript, в ній було додано підтримку структурних типів [36]. Розглянемо

наступний приклад. Функція iTakePoint2D буде приймати додаткові параметри, навіть якщо це не визначено явно (рисунок 2.17.).

```
interface Point2D {
    x: number;
    y: number;
}
interface Point3D {
    x: number;
    y: number;
    z: number;
}
var point2D: Point2D = { x: 0, y: 10 }
var point3D: Point3D = { x: 0, y: 10, z: 20 }
function iTakePoint2D(point: Point2D) { /* do something */ }

iTakePoint2D(point2D); // exact match okay
iTakePoint2D(point3D); // extra information okay
iTakePoint2D({ x: 0 }); // Error: missing information `y`
```

Рисунок 2.17 — Структурні типи даних

У TypeScript також можливе одне з найбільш поширених застосувань інтерфейсів в таких мовах, як C # і Java, - явне забезпечення відповідності класу певному контракту. Приклад інтерфейсу зображено на рисунку 2.18.

```
interface ClockInterface {
    currentTime: Date;
    setTime(d: Date);
}

class Clock implements ClockInterface {
    currentTime: Date;
    setTime(d: Date) {
        this.currentTime = d;
    }
    constructor(h: number, m: number) { }
}
```

Рисунок 2.18 — Використання інтерфейсів

Висновки до розділу 2

В цьому розділі досліджувалися можливості інтеграції СКВ, актуальність використання яких було обґрунтовано у першому розділі роботи із системою управління проектами Office 365.

В результаті дослідження було виявлено відсутність вбудованих програмних засобів для інтеграції проектів із сучасними системами хостингу репозиторіїв для СКВ хмарного пакету послуг Office 365, що значно звужує можливості при управлінні розробкою програмних продуктів. Запропонований програмний продукт дозволяє вирішити цю проблему.

Також у даному розділі було розглянуто та обґрунтовано вибір технологій, мов програмування та засобів для розробки системи, які можуть бути використаними при розробці програмного продукту.

Так як Microsoft Sharepoint висуває певні вимоги до розроблюваних додатків, а саме те, що вони повинні бути виконані у вигляді окремої HTML сторінки, з

вкрапленнями коду Javascript для реалізації проекту було обрано архітектуру односторінкових застосунків.

Для реалізації користувацької частини додатку було використано фреймворк Vue.js через його швидкодію та простоту розробки. Для обміну даними із зовнішніми використовувався текстовий формат даних JSON.

В результаті аналізу структури даних та відмінностей у реалізації REST-інтерфейсів провайдерів Github, Gitlab, Bitbucket було вирішено виділити код для взаємодії із системами хостингу СКВ в окремий шар системи, реалізований у вигляді класів мови Typescript.

3. ОПИС ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ БАЗОЮ ПРОГРАМНОГО КОДУ НА ПЛАТФОРМІ OFFICE 365

Для спрощення розробки користувацького інтерфейсу було розроблено окремий компонент-адаптер, що визначає єдиний інтерфейс незалежно від структури джерела даних. Також було реалізовано окремі клієнтські класи, що реалізують цей інтерфейс та взаємодіють із REST API потрібного сервісу. На рисунку 3.1. зображена архітектура проекту.

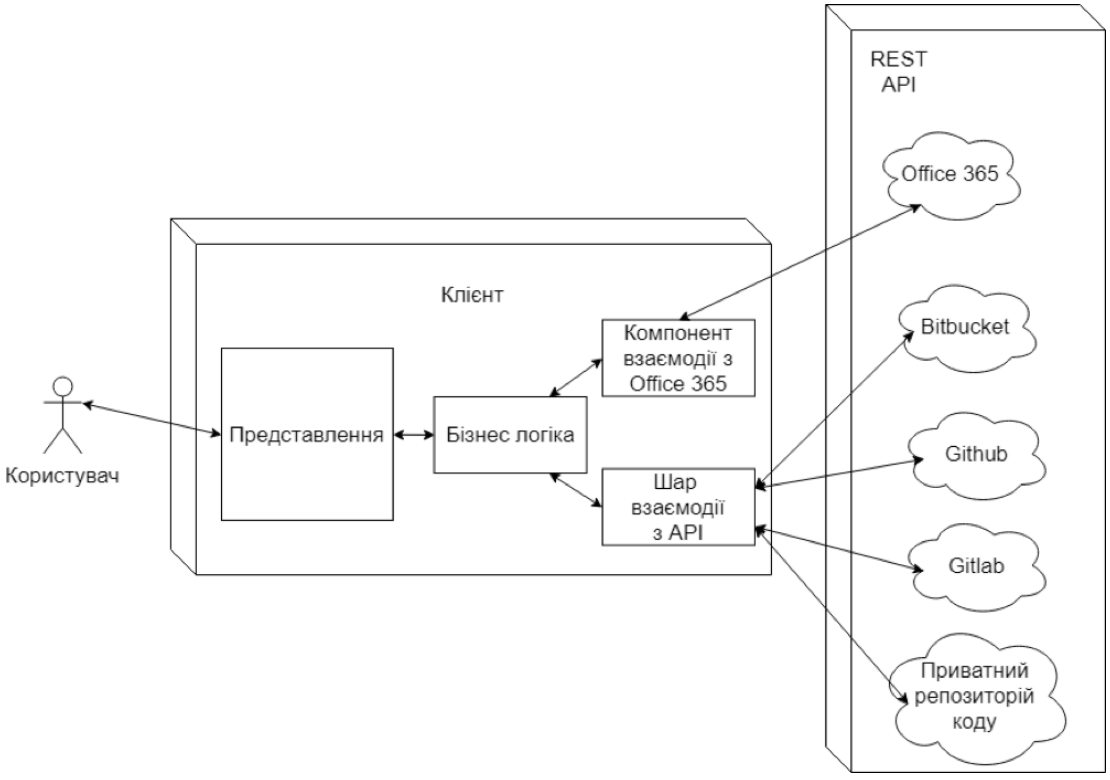


Рисунок 3.1 — Архітектура проекту

3.1 Опис програмного продукту

Для вирішення проблеми інтеграції із віддаленими репозиторіями систем керування версій було розроблено систему керування базою програмного коду у

вигляді веб-застосунку (SharePoint WebParts). Перевагою використання WebParts є можливість розробки системи без використання центрального серверу, із застосуванням технології односторінкових застосунків (single-page application).

Розроблена система дозволяє пов'язати конкретні проекти в системі Microsoft SharePoint з репозиторієм коду та підвищити рівень і якість взаємодії розробників у команді, зменшити питомі витрати на синхронізацію цих змін з іншими, суміжними проектами.

Таким чином, можна виділити основні функції системи керування базою програмного коду на платформі Office 365:

—можливість роботи з віддаленим репозиторіями програмного коду;

—підтримка різних систем контролю версій (git, tfs, svn);

—використання прошарку що дозволить абстрагуватися від реалізації окремого репозиторію або системи контролю версій.

Характерною особливістю системи є можливість вибору різних централізованих сховищ програмного коду(Github, Gitlab або Bitbucket) як джерела даних.

3.2 Структура програмного забезпечення

Основна розробка програмного продукту поділялася на 2 частини: організація структури джерела даних та розробка програмних модулів для роботи з різними джерелами даних.

Розроблений модуль веб-застосунку Sharepoint WebPart представляє собою директорію з файлами програмного коду, що має наступну структуру (рисунок 3.2.)

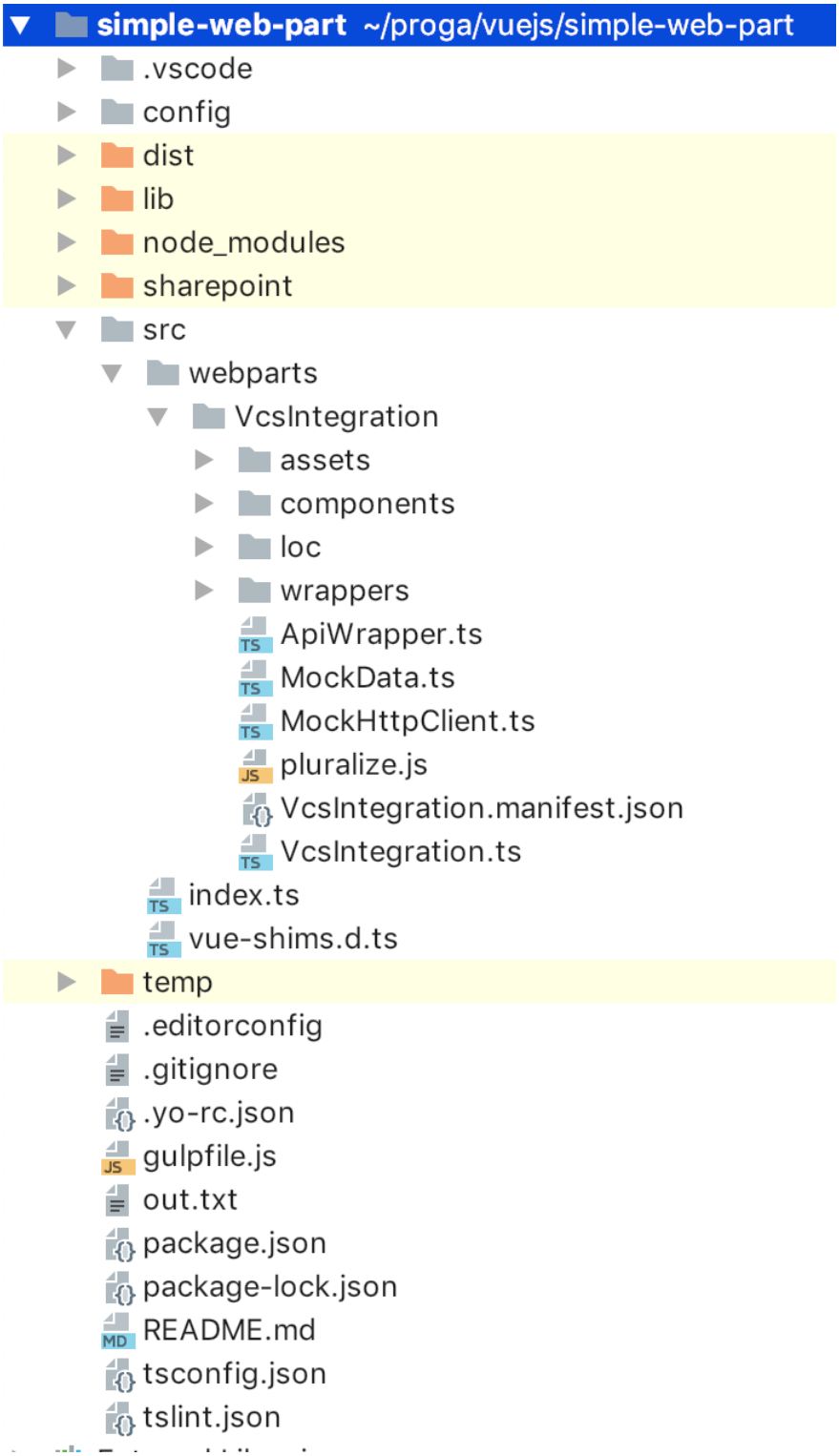


Рисунок 3.2 — Структура проекту

Структура проекту включає такі модулі:

— config – файли конфігурації для систем збірки webpack;

— node_modules – директорія з вихідним кодом необхідних зовнішніх бібілотек;

— dist, lib, temp – тимчасові файли та внутрішні бібліотеки ;

— sharepoint – файли рішення (solution), оформлені у вигляді пакунку, що може бути встановлений у системі Sharepoint;
— src/webparts – в цій директорії містяться вихідний код самої програми
  - wrappers – класи що реалізують єдиний інтерфейс, визначений у файлі types.ts надаючи додатковий шар абстракції при роботі з програмними інтерфейсами систем контролю версій Github, Gitlab, Bitbucket. Це дозволяє абстрагуватися від реалізації конкретного сервісу, при роботі з ним у компонентах Vue.js;
  - components – вихідний код компонентів Vue.js;
  - assets – містить статичні файли, такі як зображення;
  - loc – файли локалізацій, що дозволяють відображати інтерфейс на різних мовах в залежності від уподобань користувача;
  - VcsIntegration.manifest.json – файл маніфесту, в ньому визначається назва та версія застосунку, а також іконка, яку бачить користувач;
  - MockData.ts – макети об'єктів, що імітують відповіді API, та використовуються при розробці та тестуванні програмного продукту в середовищі розробки;
— package.json, package-lock.json – назви зовнішніх бібліотек та їх встановленних версій;
— tsconfig.json – конфігурація препроцесору мови TypeScript;

3.3 Опис бази даних

У якості джерела даних використовувалися списки SharePoint, що за своєю сутністю є дуже схожими на таблиці у базах даних.

Дані зберігаються у п'яти списках: «Лабораторії», «Завдання», «Проекти», «Користувачі» та «Репозиторії». На рисунку 3.3 зображено концептуальну модель предметної області

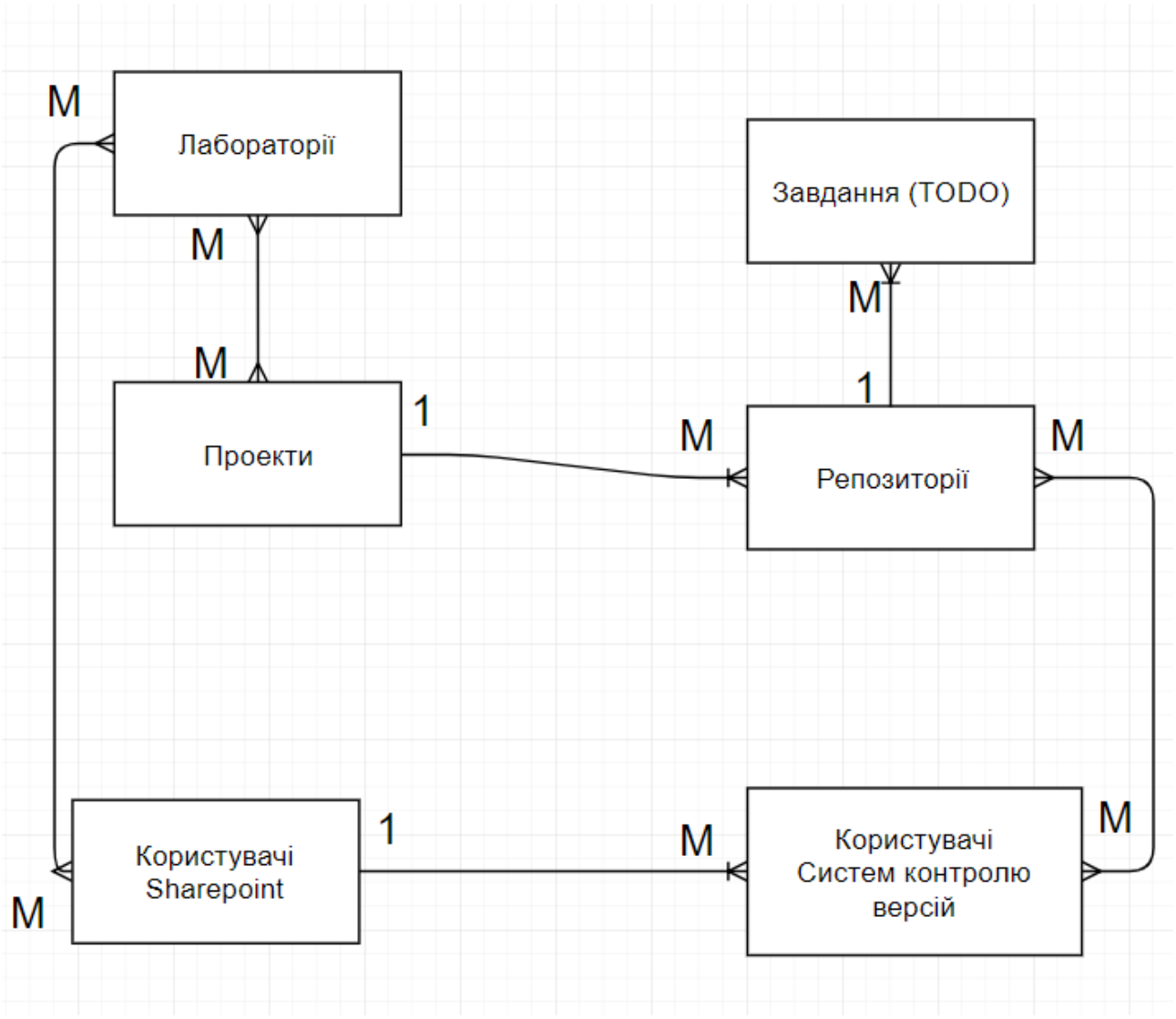


Рисунок 3.3 — Концептуальна модель предметної області

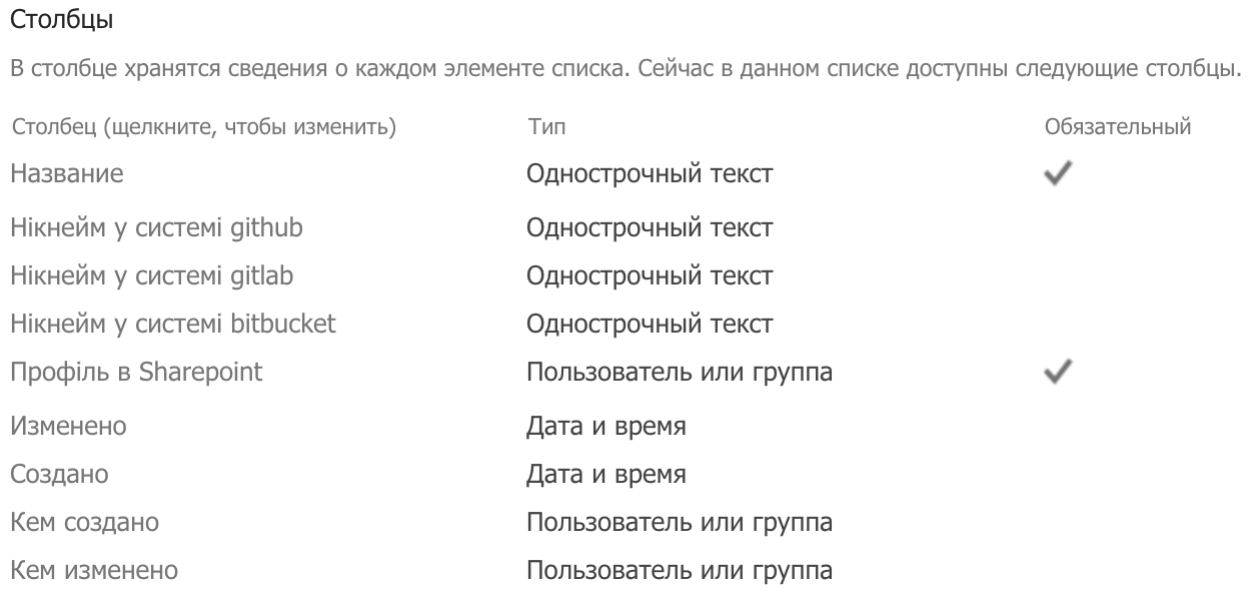
Структура списків зображена на рисунках 3.4., 3.5. та 3.6.

Столбцы

В столбце хранятся сведения о каждом элементе списка. Сейчас в данном списке доступны следующие столбцы.

Столбец (щелкните, чтобы изменить)	Тип	Обязательный
Название	Однострочный текст	✓
Робоча сторінка	Гиперссылка или рисунок	✓
Изменено	Дата и время	
Создано	Дата и время	
Кем создано	Пользователь или группа	
Кем изменено	Пользователь или группа	

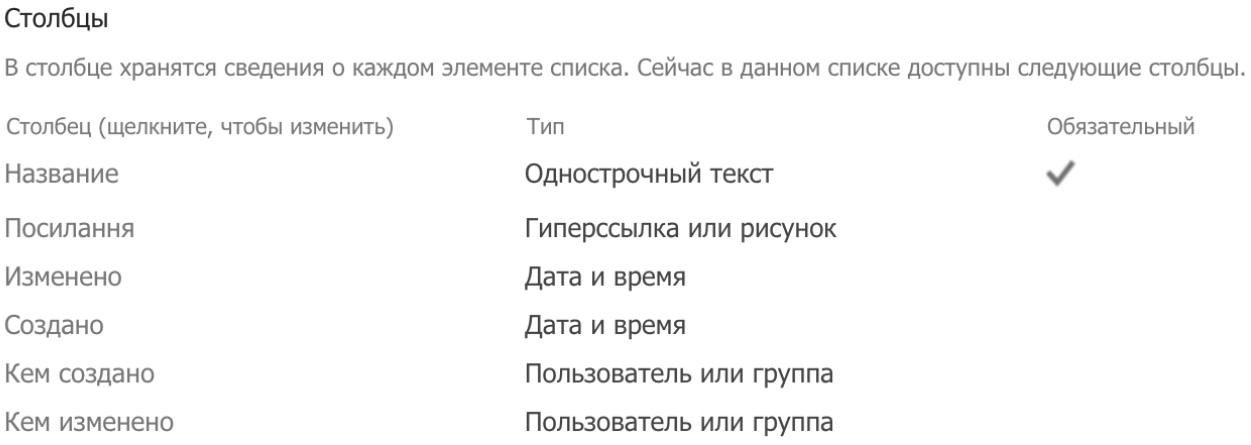
Рисунок 3.4 — Структура списку «Проекти»


Столбцы

В столбце хранятся сведения о каждом элементе списка. Сейчас в данном списке доступны следующие столбцы.

Столбец (щелкните, чтобы изменить)	Тип	Обязательный
Название	Однострочный текст	✓
Нікнейм у системі github	Однострочный текст	
Нікнейм у системі gitlab	Однострочный текст	
Нікнейм у системі bitbucket	Однострочный текст	
Профіль в Sharepoint	Пользователь или группа	✓
Изменено	Дата и время	
Создано	Дата и время	
Кем создано	Пользователь или группа	
Кем изменено	Пользователь или группа	

Рисунок 3.5 — Структура списку «Користувачі»

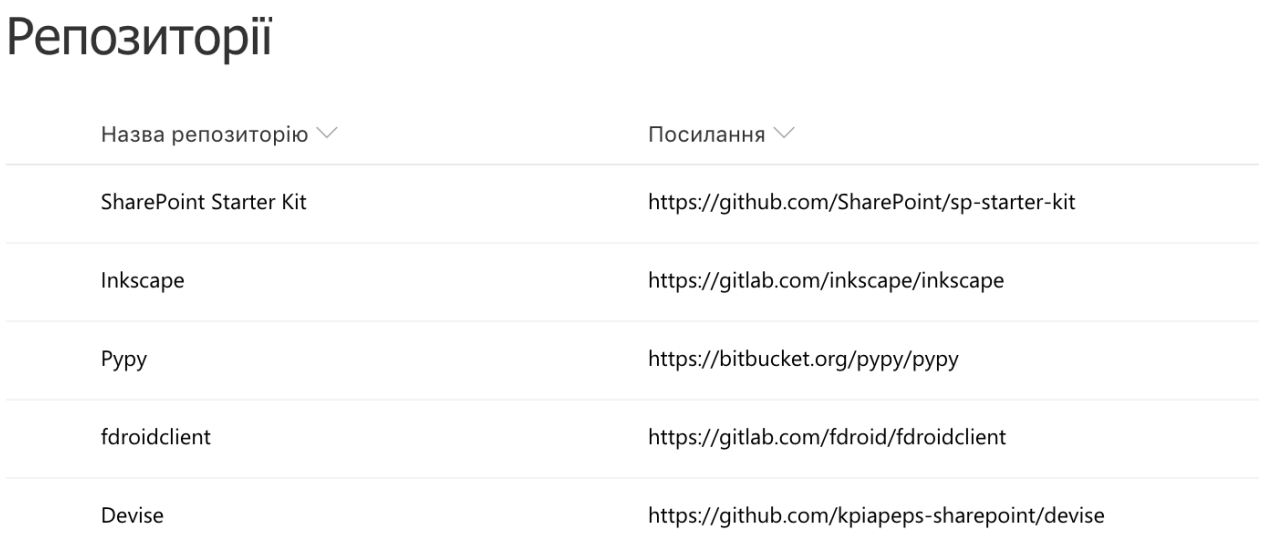

Столбцы

В столбце хранятся сведения о каждом элементе списка. Сейчас в данном списке доступны следующие столбцы.

Столбец (щелкните, чтобы изменить)	Тип	Обязательный
Название	Однострочный текст	✓
Посилання	Гиперссылка или рисунок	
Изменено	Дата и время	
Создано	Дата и время	
Кем создано	Пользователь или группа	
Кем изменено	Пользователь или группа	

Рисунок 3.6 — Структура списку «Репозиторії»

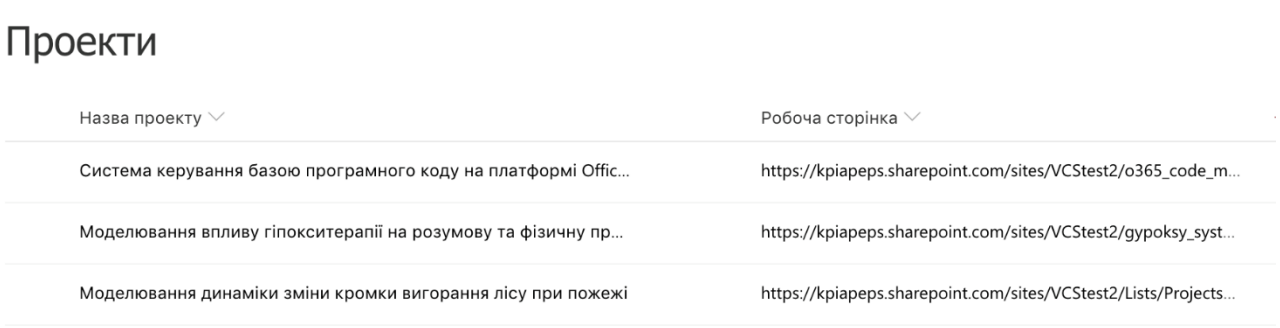
Приклади заповнення таблиць з тестовими даними зображені на рисунках 3.7., 3.8., 3.9.



Репозиторії

Назва репозиторію	Посилання
SharePoint Starter Kit	https://github.com/SharePoint/sp-starter-kit
Inkscape	https://gitlab.com/inkscape/inkscape
Pypy	https://bitbucket.org/pypy/pypy
fdroidclient	https://gitlab.com/fdroid/fdroidclient
Devise	https://github.com/kpiapeps-sharepoint/devise

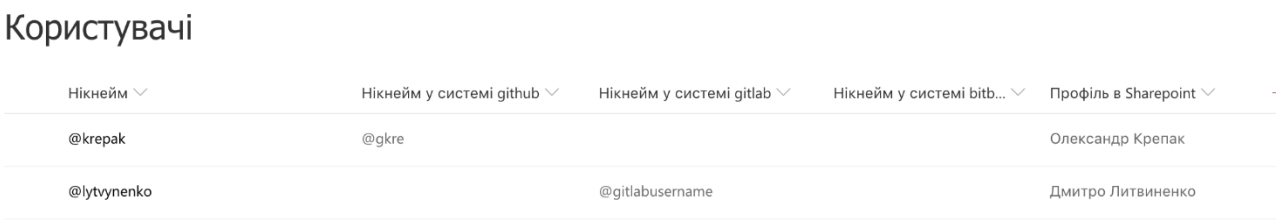
Рисунок 3.7 — Приклад заповнення списку «Репозиторії»



Проекти

Назва проекту	Робоча сторінка
Система керування базою програмного коду на платформі Offic...	https://kpiapeps.sharepoint.com/sites/VCStest2/o365_code_m...
Моделювання впливу гіпокситерапії на розумову та фізичну пр...	https://kpiapeps.sharepoint.com/sites/VCStest2/gypoksy_syst...
Моделювання динаміки зміни кромки вигорання лісу при пожежі	https://kpiapeps.sharepoint.com/sites/VCStest2/Lists/Projects...

Рисунок 3.8 — Приклад заповнення списку «Проекти»



Користувачі

Нікнейм	Нікнейм у системі github	Нікнейм у системі gitlab	Нікнейм у системі bitb...	Профіль в Sharepoint
@krepak	@gkre			Олександр Крепак
@lytvynenko		@gitlabusername		Дмитро Литвиненко

Рисунок 3.9 — Приклад заповнення списку «Користувачі»

Взаємодія із зовнішніми сервісами відбувається з використанням програмного інтерфейсу цих сервісів. Для кожного провайдеру репозиторіїв: Github, Gitlab та Bitbucket розроблено класи що реалізують цей інтерфейс. На рисунку 3.10. зображено концептуальна модель даних, з якою працюють розроблені класи.

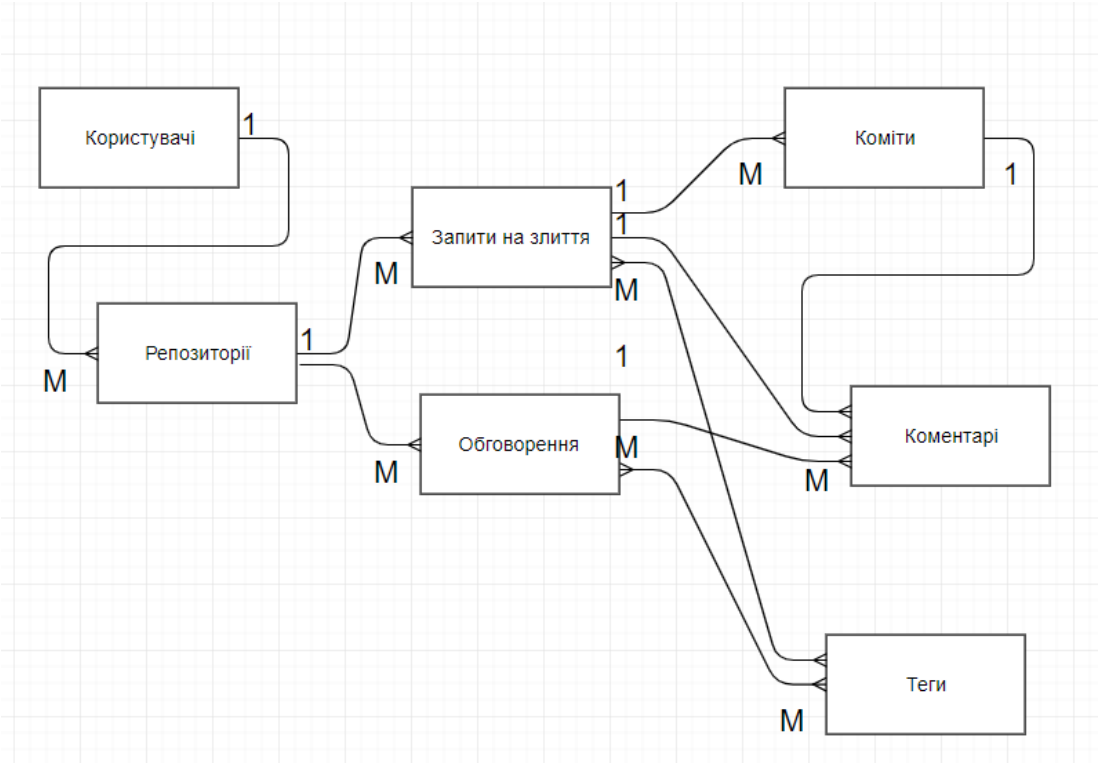


Рисунок 3.10 — Концептуальна модель даних систем контролю версій

3.4 Архітектура системи

Система побудована на принципах Model-View-View-Model (рисунок 3.11.). Хоч Vue і не реалізує патерн MVVM повною мірою, Архітектура фреймворку їм багато в чому натхненна. [37]

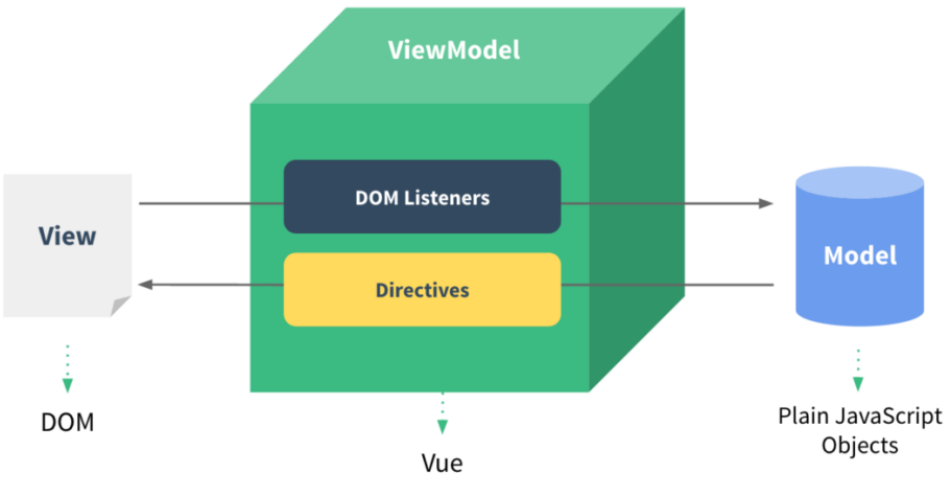


Рисунок 3.11 — Архітектура MVVM

MVVM (Model View ViewModel) - це архітектурний шаблон, заснований на MVC і MVP, який намагається більш чітко відокремити розробку інтерфейсів користувача (UI) від бізнес-логіки і поведінки в додатку. З цією метою в багатьох реалізаціях цього шаблону використовуються декларативні прив'язки даних, що дозволяють відокремити роботу з представленням від інших шарів. Це спрощує процес розробки та підтримки користувацького інтерфейсу. Розробники інтерфейсу пишуть прив'язки до ViewModel в мові розмітки документа (HTML), при цьому модель і ViewModel підтримуються розробниками, які працюють над логікою програми. Розглянемо три компоненти, що є складовими MVVM.

Модель – в MVVM представляє дані або відомості, що відносяться до конкретної області, з якими буде працювати наш додаток. Типовим прикладом даних домену може бути обліковий запис користувача (що містить в собі певні поля: ім'я, аватар, e-mail) або музичний трек (назва, рік, альбом). Моделі містять інформацію, але зазвичай не відповідають за її поведінку. Вони не займаються форматуванням інформації і не впливають на те, як дані відображаються в браузері, оскільки це не є їх зоною відповідальність. Замість цього форматування даних виконується представленням, тоді як поведінка даних вважається бізнес-логікою, яка повинна бути інкапсульована в інший шар, який взаємодіє з моделлю – ViewModel. Єдиним винятком із цього правила, як правило, є перевірка вхідних даних, тобто даних, які використовуються для визначення або оновлення існуючих моделей.

Представлення – та частина програми, з якою користувачі взаємодіють. Представлення MVVM являють собою інтерактивний користувальницький інтерфейс, що представляє поточний стан ViewModel. Представлення не несе відповідальності за оновлення даних - воно автоматично відображає всі зміни з ViewModel. Представлення в контексті Vue.js - це просто документ HTML з декларативними прив'язками, що зв'язують його з ViewModel. Представлення Vue.js відображають інформацію з ViewModel, передають йому команди (наприклад клік по елементу інтерфейсу) і оновлюється зі зміною стану в ViewModel.

ViewModel можна розглядати як спеціалізований контролер, який виконує функції конвертеру даних. Він змінює інформацію про моделі в інформацію про подання, передаючи команди з подання в модель. ViewModel пов'язує модель і відображення за допомогою подій. Якщо щось змінилося в моделі, то потрібно змінити представлення і навпаки. В Vue.js ViewModel відповідає за встановлення на видалення обробників подій в DOM, та роботу директив.

3.5 Системні вимоги

Розроблений програмний продукт потребує активного порталу Microsoft Office 365 з підтримкою Sharepoint Online, у разі використання хмарного хостингу, або окремого серверу чи ферми з серверів зі встановленою системою SharePoint Server 2016.

В таблиці 4.1 вказані вимоги до апаратного забезпечення для розгортання SharePoint Server 2016 [38].

Таблиця 3.1. Вимоги до апаратного забезпечення серверу

Сценарій установки	**Тип і масштаб розгортання**	**ОЗП**	**Місце на жорсткому диску**
Одна роль сервера, що використовує SQL Server	Розробка або встановлення ознайомчої версії SharePoint Server 2016 з мінімальним рекомендованим набором служб для середовищ розробки. Використовуйте роль односерверної ферми, за допомогою якої ви зможете вибрати, які програми-служби необхідно підготувати.	16 ГБ	80 ГБ для системного диска 100 ГБ для другого диска

Одна роль сервера, що використовує SQL Server	Установка версії SharePoint Server 2016 для пілотного або приймального тестування, в якій запущені всі доступні служби для середовищ розробки.	24 ГБ	80 ГБ для системного диска 100 ГБ для другого диска і додаткових дисків
Сервер SharePoint у фермі з кількома серверами	Установка SharePoint Server 2016 для розробки або оцінки з мінімальним набором служб.	12 ГБ	80 ГБ для системного диска 100 ГБ для другого диска і додаткових дисків
Сервер SharePoint у фермі з кількома серверами	Розгортання SharePoint Server 2016 для пілотного або приймального тестування або роботи, в якому виконуються всі доступні служби.	16 ГБ	80 ГБ для системного диска 100 ГБ для другого диска і додаткових дисків

Нижче перелічені мінімальні вимоги до програмного забезпечення для розгортання SharePoint Server 2016

Один з наступних серверів баз даних:

— 64-розрядний випуск Microsoft SQL Server 2014 SP1 (SP1);
— Microsoft SQL Server 2016 RTM;
— Microsoft SQL Server 2017 RTM для Windows.

Одна з наступних серверних операційних систем:

— Windows Server 2012 R2 Standard або Datacenter;
— Windows Server 2016 Standard або Datacenter;
— Windows Server 2019 Standard або Datacenter.

Мінімальні вимоги для клієнтських комп'ютерів: підтримуваний браузер та підключення до інтернету. Нижче вказані вимоги до програмного забезпечення комп’ютеру користувача системи (таблиця 3.2).

Таблиця 3.2. Вимоги до програмного забезпечення

Розділ	Назва
Операційна система	— Windows XP; — Windows Vista; — Windows 7; — Windows 8; — Windows 8.1; — Windows 10; — MacOS 10.12; — Linux.
Браузер (для отримання веб-сповіщень)	— Google Chrome 50 та вище; — Mozilla Firefox 44 та вище; — Opera 37 та вище; — Internet Explorer 11; — Microsoft Edge 41 та вище.

3.6 Сценарій роботи користувача з програмою

На рисунку 3.12. зображено основні сторінки додатку, та схему переходів між ними.

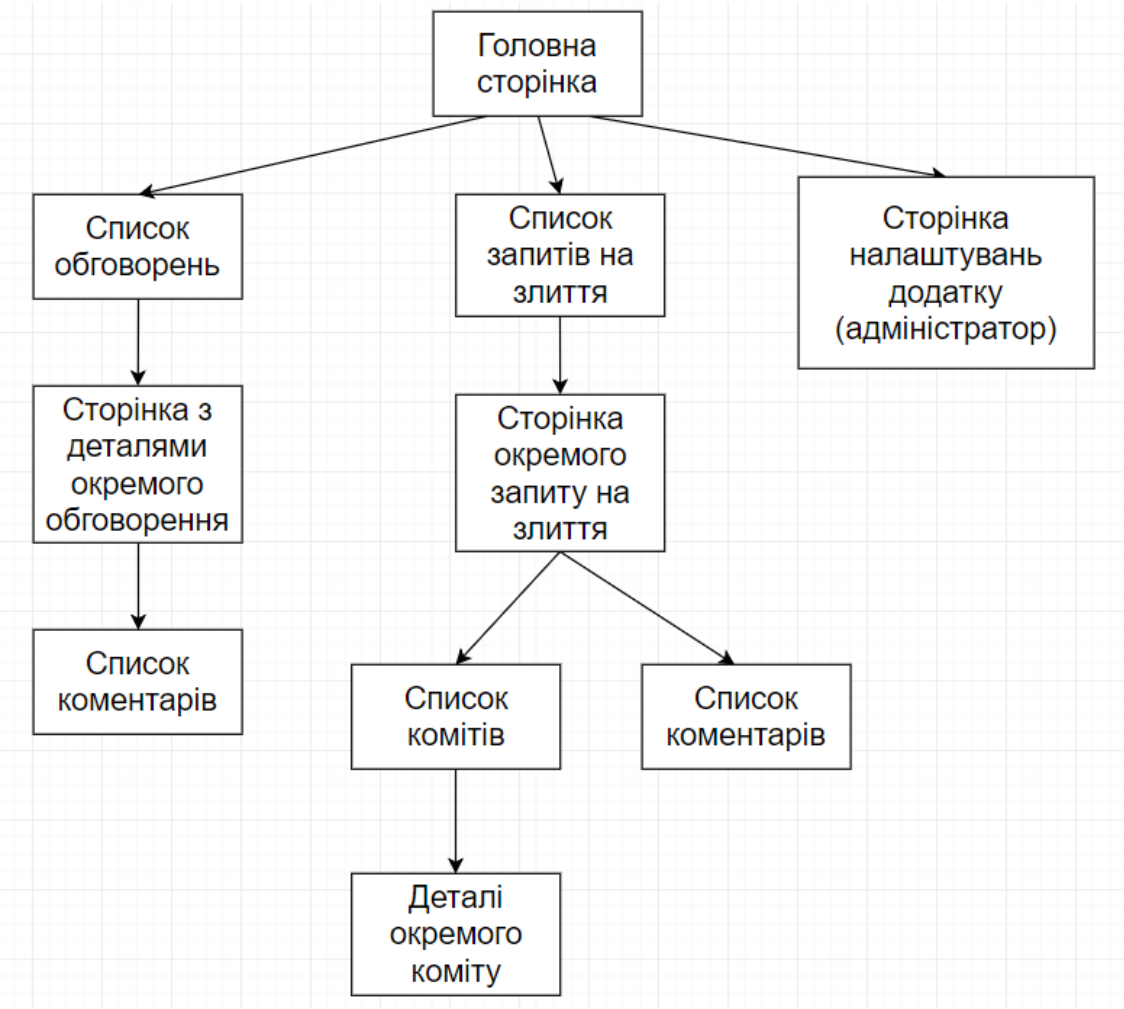


Рисунок 3.12 — Структурна схема навігації між сторінками додатку

Так як програмний продукт було виконано у вигляді компоненту системи SharePoint WebPart для використання він спочатку повинен бути налаштований адміністратором, чи іншим користувачем, що має доступ до створення або редагування сторінок на певному підсайті системи SharePoint. Для того щоб додати компонент до сторінки необхідно натиснути на кнопку  після цього з'явиться випадаюче меню, що зображене на рисунку 3.13.

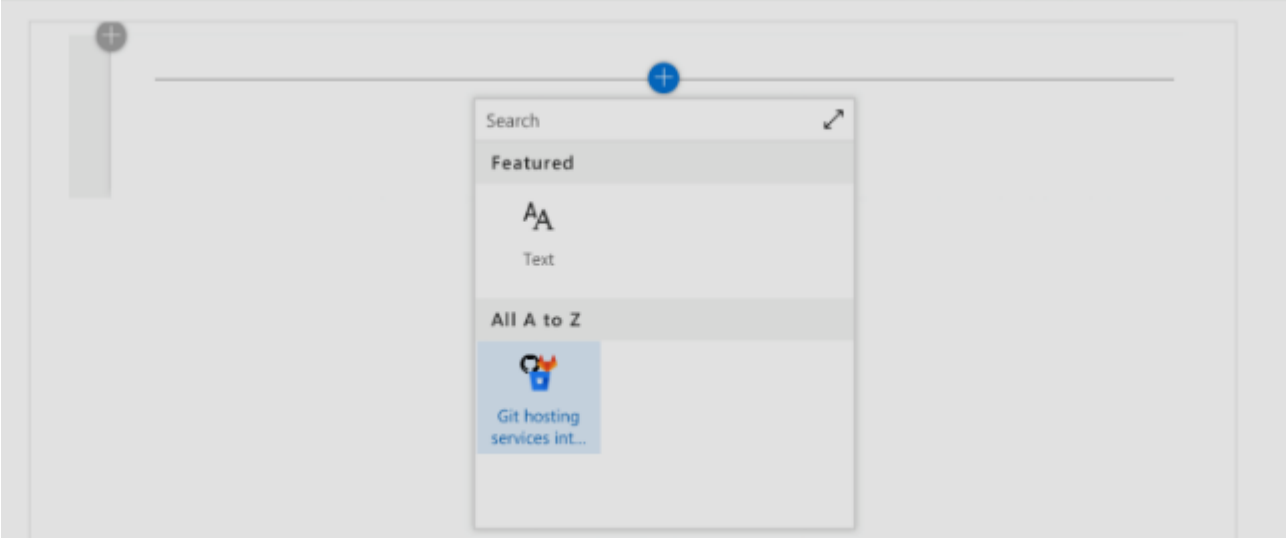


Рисунок 3.13 — Випадаюче меню для додавання компоненту

Для налаштування доданого компоненту необхідно натиснути кнопку  , що з'являється зліва від робочої області сторінки. Він також має можливість змінити положення блоку та видалити його зі сторінки, натиснувши відповідно кнопки ( та ). Через те що веб сторінка може бути відображена на різних пристроях, існує можливість імітації різних розмірів екрану, це можна зробити з панелі меню, зображеної на рисунку 3.14. Також ця панель дозволяє зберігати або відміняти дії та переглядати дані веб-застосунку, що будуть згенеровано в результаті використання інтерфейсу для налаштування сторінок.

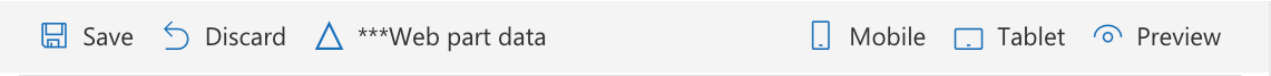


Рисунок 3.14 — Панель меню інтерфейсу для налаштування сторінок

Користувач може налаштувати деякі опції компоненту, а саме: вибрати репозиторій з випадаючого списку, для якого буде відображатися інформація а також максимальну кількість елементів що будуть відображатися на створеній сторінці. Панель конфігурації зображена на рисунку 3.15.

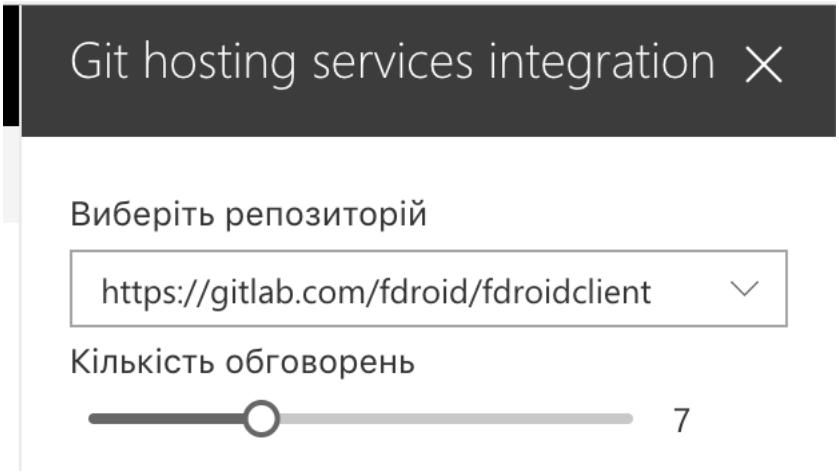


Рисунок 3.15 — Панель конфігурації веб-компоненту

Кінцевий користувач не матиме доступу до редагування сторінки а лише зможе її переглядати. При відкритті сторінки з компонентом користувач бачить вікно зі списком останніх обговорень, що зображено на рисунку 3.16.

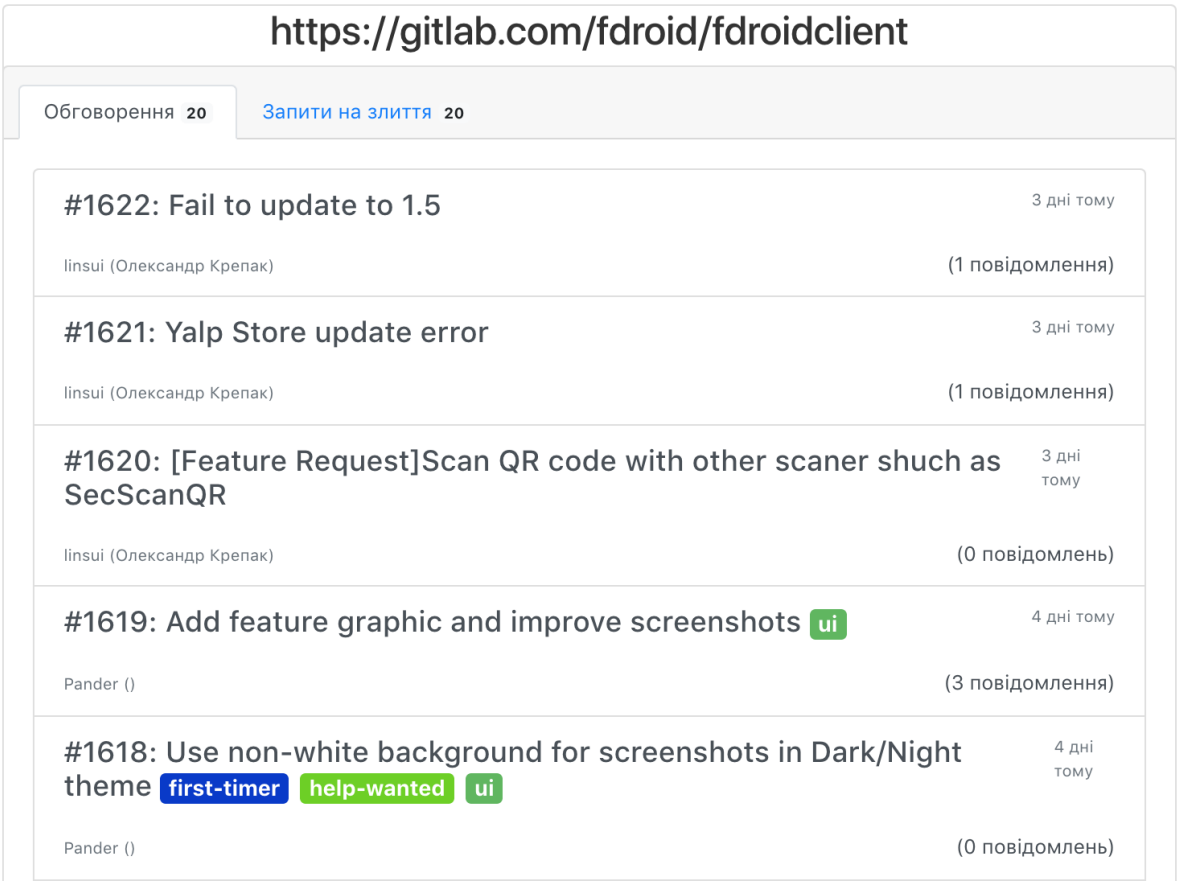


Рисунок 3.16 — Вікно списку останніх обговорень

В верхній частині відображається адреса репозиторію для якого показуються дані та дві вкладки що дозволяють переглядати останні обговорення та запити на злиття. В елементах списку відображається тема або назва обговорення, теги що присвоєні обговоренню, інформація про те як давно обговорення було створено, нікнейм користувача що її створив та ім'я користувача SharePoint, що пов'язане з цим нікнеймом. Для перегляду деталей та коментарів до відповідної теми обговорення користувачу необхідно натиснути лівою кнопкою миші на відповідний елемент списку. На рисунку 3.17 зображено область сторінки у режимі детального перегляду обговорення.

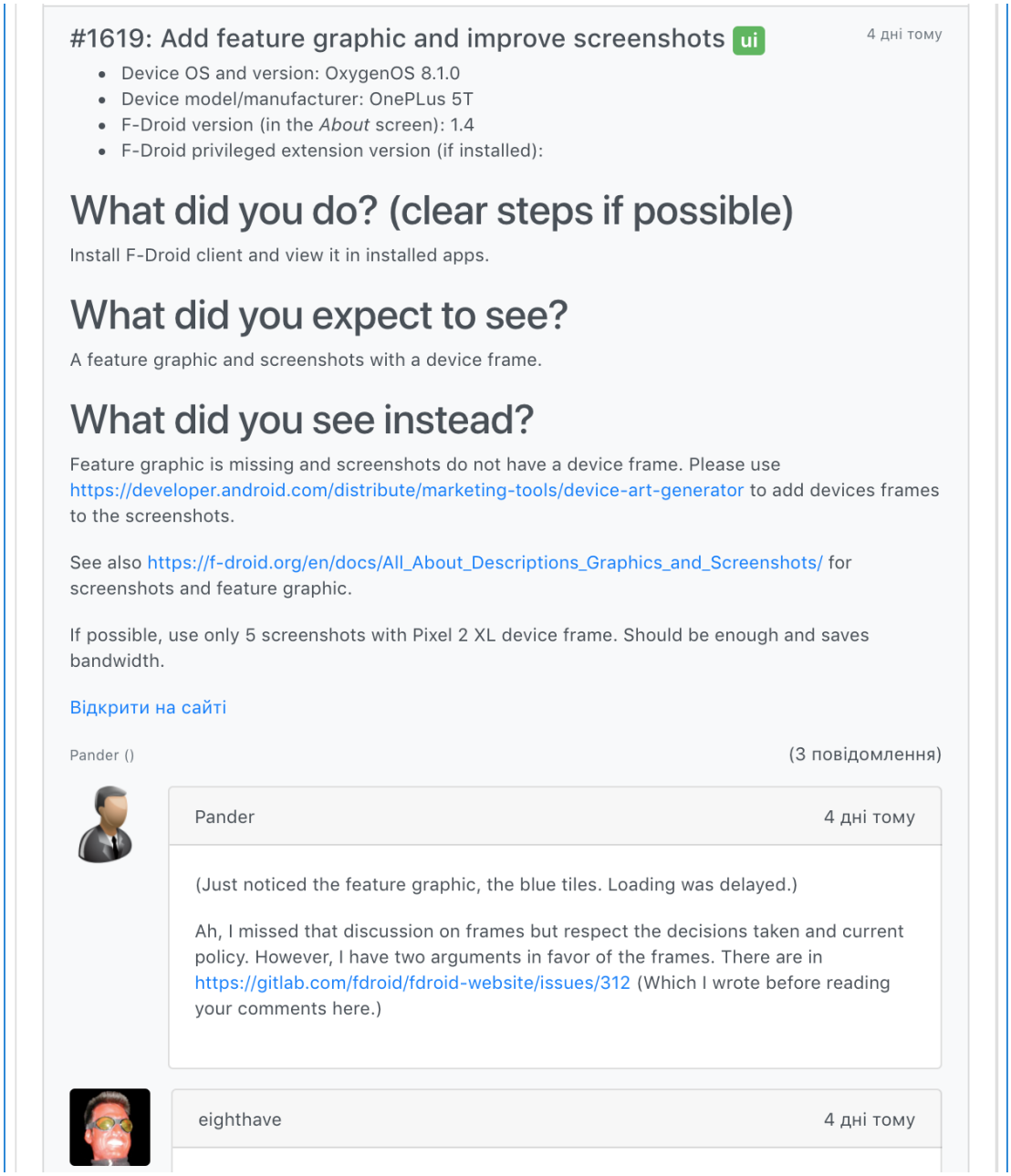


Рисунок 3.17 —сторінка у режимі детального перегляду обговорення.

У розгорнутому вигляді відображається текст першого повідомлення теми та коментарі інших користувачів до нього. Користувач має можливість перейти на сайт репозиторію з цим обговоренням, натиснувши на посилання "Відкрити на сайті".

На рисунку 3.18 зображено інтерфейс при перегляді деталей на злиття. За структурою вони нагадують деталі обговорень, тобто користувачі мають можливість додавати коментарі, щодо доданих автором змін. Користувач також бачить список пов'язаних комітів з їх описом та посиланням на список змін, які можна переглянути на сайті, що здійснює хостинг цього репозиторію. Кожен користувач, що має доступ на перегляд запитів на злиття може приймати внесені зміни або відхиляти запити на зміни, із коментарем що треба змінити у разі якщо запропоновані призводять до неправильної роботи програми або не відповідають вимогам до стилю розробки конкретної команди. Після цього один з адміністраторів репозиторію переглядає ці зміни та коментарі інших користувачів, та на їх основі приймає рішення про остаточне прийняття цих змін.

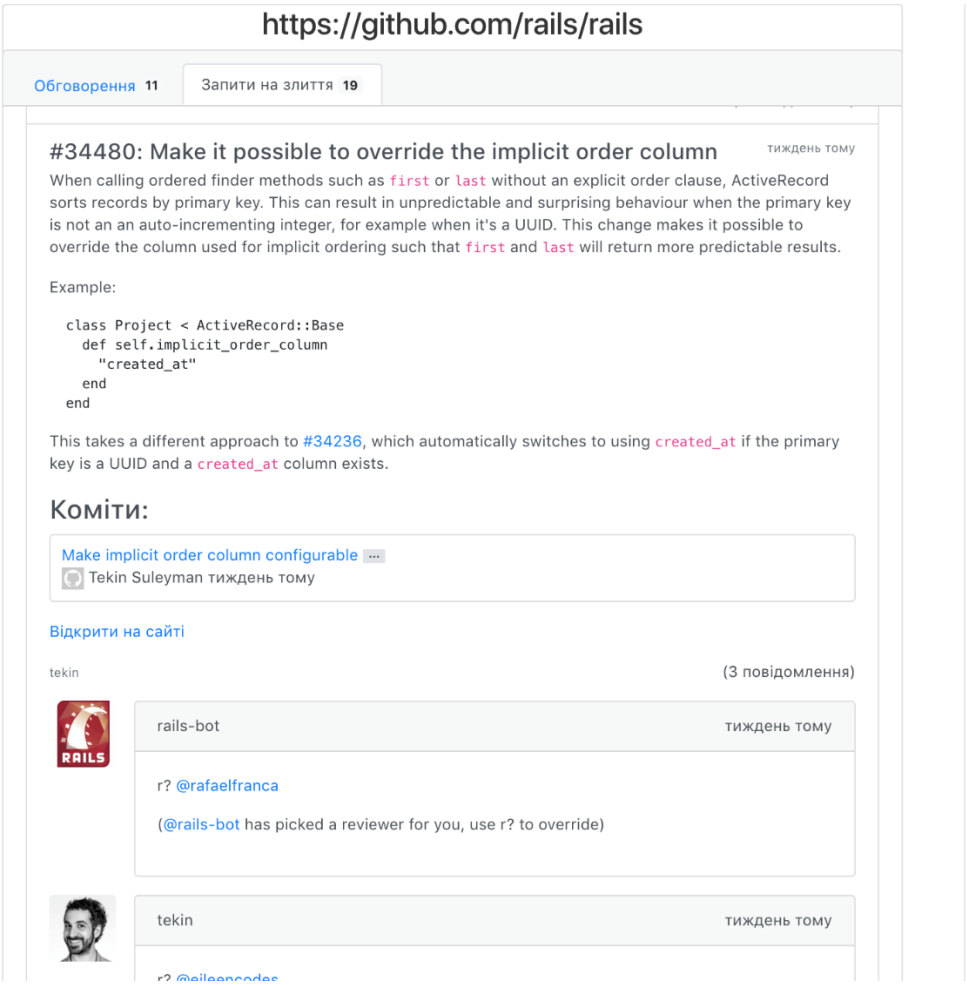


Рисунок 3.18 — область сторінки у режимі запитів на злиття.

Висновки до розділу 3

Так як інтеграція СКВ із окремими веб-порталами Sharepoint передбачає збереження адрес репозиторії та списку на основі якого можна пов'язати користувачів СКВ та користувачів Office 365 постає необхідність використання централізованої бази даних.

У якості бази даних використовуються списки Sharepoint, що надають більшість можливостей реляційних баз даних. Дані зберігаються у п'яти списках: «Лабораторії», «Завдання», «Проекти», «Користувачі» та «Репозиторії».

На основі вибраних технологій, визначених в розділі було реалізовано програмний додаток, обґрунтовано використання архітектури Model-View-View-Model для організації роботи користувацького інтерфейсу. Структура додатку складається з окремих компонентів Vue.js, що взаємодіють між собою через інтерфейс вхідних параметрів.

Користувацький інтерфейс довзоляє переглядати обговорення та запити на злиття а також коментарі користувачів до них, що зберігаються на сервері систем контролю версій, а також пов'язує нікнейми користувачів системи контролю версій із іменами користувачів системи Sharepoint.

Вікно налаштування дозволяє адміністратору налаштувати додаток, вибравши посилання на репозиторій, для якого буде показана інформація, та обрати кількість обговорень для відображення. Інтерфейс також передбачає можливість перегляду списку пов'язаних комітів, коментарів до них та аналізу внесених змін, внесених у файли.

4. СТАРТАП-ПРОЕКТ

Розділ має на меті проведення маркетингового аналізу стартап-проекту задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження. Проведення маркетингового аналізу передбачає виконання нижченаведених кроків.

4.1 Резюме проекту

Зміст ідеї проекту полягає в розробці системи керування базою програмного коду у вигляді веб-застосунку SharePoint WebParts, що дозволяє розгортати систему без використання центрального серверу.

Ціннісна пропозиція полягає в вирішенні проблему організації взаємодії при розробці програмного продукту. Продукт задовольняє потребу інтеграції системи керування версіями з внутрішніми даними системи та використання SharePoint у якості системи керування версіями

Основними споживачами є розробники програмного забезпечення та керівники проектів.

Конкурентними перевагами проекту є

— відсутність альтернатив;

— нова технологія;

— зручність використання;

— сформована клієнтська база.

Інвестиційна привабливість проекту у його унікальності та фактично відсутності реальних конкурентів, а також можливості легко вийти на інші, міжнародні ринки з мінімумом витрат.

4.2 Організація проекту

Даний проект реалізовується на теоретичній та технологічній базі НТУУ «КПІ ім. І. Сікорського». Період створення – 05.10.2017 – 30.12.2018. НТУУ «КПІ ім. І. Сікорського» має досвід в розробці складних інформаційних систем керування завданнями та їх практичного використання. Проект направлений на розширення та оновлення існуючої технології організації взаємодії при розробці програмного продукту великою групою розробників.

4.3 Канва бізнес-моделі проекту

Так звана канва бізнес-моделі, запропонована Алексом Остарвальдером, представляє собою інструмент для швидкої оцінки ситуації, що звільняє від необхідності написання бізнес планів [39].

Таблиця 4.1. Канва бізнес-моделі проекту

<table>
<tr>
<td rowspan="2">(8) КЛЮЧОВІ ПАРТНЕРИ
Партнер - НТУУ «КПІ»
Постачальник - НТУУ «КПІ»
Необхідні ресурси: теоретичний матеріал, програмне та апаратне забезпечення, данні що зберігаються в системі
Ключова діяльність якою займаються партнери – підговка кваліфікованих кадрів</td>
<td>(4) КЛЮЧОВІ ВИДИ ДІЯЛЬНОСТІ
Керування задачами та контроль ступіню виконання проекту</td>
<td rowspan="2">(5) ЦІННІСНІ ПРОПОЗИЦІЇ
Вирішує проблему організації взаємодії при розробці програмного продукту
Задовільняє потребу інтеграції системи керування версіями з внутрішніми даними системи та використання SharePoint у якості системи керування версіями</td>
<td>(6) ВЗАЄМОВІДНОСИНИ ЗІ СПОЖИВАЧАМИ
Колективна взаємодія між споживачами, працівник використовує програмне забезпечення з робочого ПК</td>
<td rowspan="2">(5) СПОЖИВЧІ СЕГМЕНТИ
Споживачі – організації-користувачі програмної пакет послуг Microsoft Office 365 які потребують інтеграції системи керування версіями програмного коду із внутрішньою документацією.
Наприклад, сфера оборони, прогнозування, тощо.</td>
</tr>
<tr>
<td>(6) КЛЮЧОВІ РЕСУРСИ
Основними ресурсами є дані що зберігаються в системі</td>
<td>(6) КАНАЛИ ЗБУТУ
Каналами збуту можуть бути: веб-сайти, організації та компанії, навчальні установи</td>
</tr>
</table>

(7) СТРУКТУРА ВИТРАТ	(8) ПОТОКИ НАДХОДЖЕННЯ ДОХОДІВ
Витрати на рекламу та маркетинг, післяпродажну підтримку та вдосконалення програмного продукту	Продукт використовує бізнес-модель продажу одноразової ліцензії та додаткової платної підтримки. Основна інвестиційна привабливість проекту у його унікальності та фактично відсутності реальних конкурентів, а також модливості легко вийти на інші, міжнародні ринки з мінімумом витрат

4.4 Ключові види діяльності проекту

4.4.1 Вид проекту за характером інновації

— дослідно-технологічна робота: аналіз схожих рішень, опис предметної області та постановка задачі, розробка нової програмної підсистеми та документації до неї;

— запровадження нової технології: розробка компонентів для єдиної системи на основі хмарного офісного пакету послуг Microsoft Office 365, що забезпечить користувачам доступ до документів і даних через браузер з будь-якого пристрою з можливістю виходу в Інтернет.

4.4.2 Спрямованість проекту

— випуск продукції, конкурентоспроможної на світовому ринку, через те що вихід на інші ринки не потребує великих витрат через поширеність системи, для якої розроблюється програмний продукт;

— збільшення продуктивності та поліпшення умов праці, дозволяє поєднати всі інструменти для управління процесом розробки програмних коду зокрема керувати та планувати задачі та призначати відповідальних.

4.4.3 Висновок щодо науково-технічного рівня ідеї

Немає аналогів в світі або краща за існуючі в світі аналоги: наразі нема програмної платформи, яка б інтегрувалася в програмну систему Microsoft Sharepoint, та надавала можливості пов'язування усього вбудованого функціоналу системи із віддаленим репозиторієм системи контролю версій. При побудові програмного продукту може використовуватись каскадна модель життєвого циклу, в ній кожен етап роботи виконується лише раз. На кожному етапі робота виконується

настільки ретельно, щоб потреби повертатись до попереднього не виникало. Спочатку визначаються функціональні вимоги до системи, на наступному етапі було обрано засоби реалізації що дозволяють реалізувати необхідну функціональність. На етапах, що пов'язані із розробкою програмного продукту дуже важливо фіксувати кожен крок та прогрес, що дозволить у разі необхідності повторно використовувати певні рішення та фіксувати побажання замовника, які будуть реалізовані, на основі цієї інформації створюватимуться звіти для керівництва та бухгалтерії, що дозволить ретельніше керувати загальним процесом розробки.

4.4.4 Основні бізнес-процеси проекту.

Виділення основних бізнес-процесів організації виконується на основі інформації про зовнішнє оточення і основних інформаційних і матеріальних потоках по принципу:

— клієнт процесу;

— продукт, що він використовує;

— основний бізнес процес організації.

Кількість основних бізнес процесів бажано обмежити (не більше 7). Крім основних, необхідно визначити допоміжні процеси. Загальна кількість процесів верхнього рівня на повинна перевищувати 13-15 пунктів [40].

Таблиця 4.2. Основні бізнес-процеси проекту

Група процесів	*Бізнес-процес*	*Ступінь опрацювання бізнес-процесу*	
		є реалізованим	*буде реалізованим*
Розробка продукції	розробка та конструювання продукції	-	-
	розробка і конструювання процесу	-	-
	технологічна підготовка виробництва	-	-
Вимоги споживачів	дослідження розвитку ринку	+	+
	організація маркетингу і продажів	+	+

	тендерне розміщення замовлень	-	-
Виконання замовлень	забезпечення і матеріально-технічний збут	+	+
	планування і управління виробництвом	-	-
	виробництво продукції	-	-
	розподіл продукції і логістика	+	+
Обслуговування споживача	післяпродажне обслуговування	+	+
	повернення продукції	-	+

4.5 Ціннісні пропозиції та споживачі

Цінністна пропозиція – сукупність переваг, які проект може запропонувати споживачу.

Для розроблення якісної цінністної пропозиції необхідно вирішити якими завданнями, проблемами та вигодами ви будете займатися, а які залишите в спокої. Жодна цінністна пропозиція не може бути орієнтована на все одразу. Якщо ваша карта цінності показує саме це, можливо, ви були не досить чесні при визначенні задач, проблем та переваг для профілю користувача [41].

4.5.1 Характер формування споживчої цінності проекту

— покращення задоволення існуючих потреб: інтеграція системи керування версіями, а також інших внутрішніх інструментів та систем, що використовуються в процесі розробки та підтримки програмних продуктів.

4.5.2 Зміст ідеї проекту

Ідея проекту та її зміст - чи не найважливіші складові успіху. Найчастіше те, з чим доводиться мати справу, не може бути прийнято керівництвом. Наприклад, не варто пропонувати Міноборони проект державно-приватного партнерства по виробництву військових безпілотників, що заснований на іноземній технології (і

саме прикладів таких звернень - кілька). Він може задовольняти будь-яким іншим критеріям і мати відмінну економіку у своїй основі, але в підсумку це безцільна трата часу. Хоча консультанти, яким ви заплатите багато грошей за стос паперу, з задоволенням її підготують, ви дізнаєтеся про всі іноземні аналоги і порівнянних моделях, отримаєте короткий екскурс в історію і прогноз світлого майбутнього, але до бажаного результату вас це аніскільки не наблизить. Також часто зустрічаються випадки, коли пропонована бізнесом ідея в чистому вигляді не годиться через те що вона є неконкурентоспроможною або її складно монетизувати [42].

Таблиця 4.3. Зміст ідеї проекту

Зміст ідеї	*Напрямки застосування*	*Сегменти споживачів*	*Цінність для споживачів*
Розробка системи керування базою програмного коду у вигляді веб-застосунку SharePoint WebParts, що дозволяє розгортати систему без використання центрального серверу	1. Інтеграція з різними системами керування версіями	Компанії, що займаються розробкою чи підтримкою програмного забезпечення	Можливість пов'язати конкретні проекти в системі Microsoft SharePoint з репозиторієм коду
	2. Розв’язання задачі керування проектами	Керівники проектів	Полегшує можливість відслідковування прогресу виконання того чи іншого проекту
	3. Підтримка панелі приладів, що відображає стан виконання проекту на даний момент(невиконані завдання, результати білдів, календар нарад та майбутніх подій)	Розробники програмного забезпечення	Спрощує взаємодію всередині команди

4.5.3 Аналіз ідеї проекту

Таблиця 4.4. Аналіз ідеї проекту

№	*Техніко-економічні характеристики ідеї*	*Продукція конкурентів*		*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Мій проект	*Кон-т 1 (вбудованні можливості Mircosoft Sharepoint)*			
1.	Затрати на підтримку	мінімальні	низькі			+
2	Затрати на реалізацію	середні	низькі	+	-	
3	Повна підтримка систем контролю версій git,svn,tfs	високі	мінімальні			+
4	Кросплатформність	+	+		+	

4.5.4 Технологічний аудит ідеї проекту

Дж. Ліндсей (J. Lindsay, 1994, p. 2) формулює наступні цілі технологічного аудиту:

— виявити і оцінити технологічні ресурси і можливості компанії;

— оцінити значимість ринку або потенціал стратегій компанії;

— оцінити конкурентну позицію компанії з точки зору використовуваних технологій;

— усвідомити і виявити способи розвитку і використання компанією технологій для формування і підтримки стійкого конкурентної переваги.

Д. Форд (D. Ford, 1988) пропонував певну схему, а також ряд питань, на які необхідно відповісти. Він також підкреслює, що такий аудит повинен стати безперервним процесом управління [43].

Таблиця 4.5. Технологічний аудит ідеї проекту

№	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Ідентифікація об'єкта, виведення можливих результатів	Мова Javascript, технологія SPA (VueJS)	+	+
2.	Використання бази даних	IndexedDB	+	+
Обрана технологія реалізації ідеї проекту: Javascript, VueJS				

4.5.5 SWOT-аналіз проекту

SWOT-аналіз— одна з відправних точок у розробці стратегії компанії. Термін SWOT — акронім, сформований з слів strengths (сильні сторони), weaknesses (слабкі сторони), opportunities (можливості) і threats (загрози). Одним ялиновому SWOT — керівництво для формування вашої думки про власну компанію і оточення, в якому вона існує. Сильні і слабкі сторони компанії-частина внутрішнього аналізу фірми. Можливості і загрози-частина зовнішнього аналізу оточення компанії, тобто всього, що може вплинути на неї ззовні.

Щоб провести невеликий SWOT-аналіз для вашої компанії, необхідно відповісти на такі питання, це дозволить визначити ваше положення на ринку.

1. У чому полягають сильні сторони компанії?

2. У чому полягають слабкі сторони компанії?

3. Які можливості відкриваються перед компанією в майбутньому?

4. Які загрози можуть перешкодити досягти поставлених цілей і як компанія планує з ними справлятися? [44]

Таблиця 4.6. Визначення сильних, слабких та нетральних характеристик

Сильні сторони (S): – *Відсутність альтернатив* – *Нова технологія* – *Зручність використання* – *Сформована клієнтська база*	*Слабкі сторони (W):* – *Невелика (відносна) кількість цільових користувачів* – *Необхідність використання/налаштування зовнішніх систем*
Можливості (O): – *Вільна ніша на ринку* – *Низька собівартість підтримки* – *Додаткові послуги*	*Загрози (T):* – *Розробник (Microsoft) може у майбутньому розширити вбудований функціонал* – *Слабкий імідж продукції* – *Ріст популярності інших систем організації командної роботи*

4.6 Взаємовідносини зі споживачами та канали збуту

Стратегія сегментації ринку дозволяє компанії, враховуючи свої сильні і слабкі сторони, вибрати ті з них, які забезпечать концентрацію ресурсів саме в тих сферах діяльності, де підприємство має максимальні переваги чи, принаймні, мінімальні недоліки.

Новаторська сегментація - головний секрет маркетингу. Лише той, хто знаходить новий погляд на ринки, здатний змінити правила н грати за межами кордонів, встановлені конкурентами. Наприклад, надання кредиту за граничними верхніми ставками, особам, які мають погану кредитну історію.

Існує два традиційних підходи до розробки стратегії сегментування:

— почати з дослідження ситуації кон'юнктури ринку в області традиційних видів товарів (послуг), виявлення фактичних і потенційних споживачів і відмінностей їх ставлення до нових видів товарів (послуг);

— почати з формування уявлення про те, які змінні характеризують той чи інший сегмент споживачів.

Основні помилки, які допускаються при позиціюванні товарів (послуг) на ринку:

— позиціювання поза ринком;
— позиціонування на ринку в цілому, без розділення його на традиційний і перспективний;
— позиціонування шляхом прямого протиставлення товарів (послуг) конкурентів. [45]

Таблиця 4.7. Визначення сегменту споживачів та каналів збуту

№	*Сегмент споживачів*	*Особливості поведінки*	*Вимоги споживачів*	*Канали збуту*	*Інші аспекти взаємовідносин*
1.	Навчальні заклади, в тому числі інститути та університети	Стабільність роботи ПЗ	Низька ціна продукту Просте та недороге обслуговування,	Держава	
2	Компанії та підприємства, що займаються розробкою ПЗ	Можливість вдосконалення та модифікації ПЗ відповідно до вимог	Відсутність проблем при інтеграції, гарантії подальшої підтримки	Директори та менеджери проектів, інші компанії	
3	Малі команди розробників	Гнучкість	Наявність демо-версії, невисока ціна	Менеджери проектів	

4.7 Обґрунтування ресурсів та витрат проекту

Оцінка перспективних витрат ресурсів у проекті виключно важлива в умовах їх обмеженої кількості. Тобто практично завжди і всюди. Виняток становлять

організації, в яких ресурси необмежені або, що буває частіше, не враховуються. Очевидно, що на початковому етапі «Визначення» точно оцінити ресурси, які потрібні на етапі реалізації майбутніх рішень, дуже складно. Отже, починати треба з оцінок і припущень, а потім, у міру отримання додаткової інформації, проводити уточнення і коригування заявлених вимог. При оцінках слід враховувати всі можливі вимоги до ресурсів: і робочий час фахівців, і модернізацію обладнання, і інші фінансові витрати. Навіть дуже неточні прогнози на перших кроках проекту потрібні для відображення поточного розуміння майбутніх витрат і ефективності проекту. У статуті проекту зазначаються тільки основні вимоги до ресурсів. Для подробиць, у разі необхідності, готується спеціальний додаток до статуту проекту з обґрунтуванням оцінок і прогнозними розрахунками. [46]

4.7.1 Визначення ціни

Таблиця 4.8. Визначення ціни

№	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на продукцію*	*Розрахункова ціна продукції*
1.	500-1000$	1000-2000$	8000$/міс	500-2000$/міс	1000$

4.7.2 Визначення обсягу виробництва продукції

Таблиця 4.9. Визначення обсягу виробництва продукції

Показник	*Значення по роках*				
	2019	*2020*	*2021*	*2022*	*2023*
Загальна потреба в продукції	100%	95%	89%	74%	51%
Можливі річні обсяги випуску в натуральних показниках	5 од.	5од.	5од.	10 од.	10 од.
Ціна одиниці продукції (тис. $.)	1000	1000	1000	1000	1000
Річні обсяги випуску в вартісних показниках (тис. $.)	5000	5000	5000	10000	10000

4.7.3 Розрахунок загальних початкових інвестиційних витрат

Таблиця 4.10. Визначення обсягу виробництва продукції

№	*Назва етапу*	*Терміни виконання*	*Обсяги фінансування, тис. грн.*
1.	Проведення досліджень	*31.11.2017-01.02.2018*	*20*
2.	Розробка або придбання технології	*01.02.2018-12.08.2018*	50
3.	Придбання устаткування	*12.08.2018-20.08.2018*	15
4.	Організація діяльності та запуск проекту	*20.08.2018-01.09.2018*	10
5.	Витрати на управління	*01.09.2018-10.10.2018*	7
6.	Початкові виробничі витрати	*12.08.2018*	20
7.	Інші витрати	*01.09.2018*	5
	Разом		*127*

4.7.4 Розрахунок виробничих витрат

Таблиця 4.11. Розрахунок виробничих витрат

№ з/п	*Стаття витрат*	*Сукупні витрати за період, тис. грн.*				
		2018	*2019*	*2020*	*2021*	*2022*
1.	Загальногосподарські витрати	14	14	18	18	18
1.1.	витрати на оренду та утримання приміщень, обладнання	4	4	4	4	4
1.2.	комунальні витрати	2	2	2	2	2
1.5.	витрати на збут, просування та рекламу	8	8	12	12	12
2.	Витрати на матеріальні ресурси (комплектуючі. сировина)	5	5	5	5	5
3.	Витрати на оплату праці	10	10	10	10	10
4.	Інші витрати (якщо є)	0	0	0	0	0
	Разом:	29	29	29	33	33

4.7.5 Розрахунок загальних витрат на реалізацію проекту по роках

Таблиця 4.12. Розрахунок загальних витрат на реалізацію проекту по роках

Показник	*Значення по роках*					*Разом*
	2018	*2019*	*2020*	*2021*	*2022*	

Інвестиційні витрати	*127*	*0*	*0*	*0*	*0*	*127*
Виробничі витрати	29	29	29	29	29	145
Обсяг загальних витрат, в тому числі за рахунок						
– *власних коштів*	0	0	0	0	0	0
– *коштів інвестора*	156	29	29	29	29	145

4.8 Грошовий потік та оцінка вартості проекту

Центральне місце в комплексі заходів по визначенню ступеня обґрунтованості інвестиційних рішень і аналізу ефективності проектів займає оцінка майбутніх грошових потоків, що виникають в результаті вкладення коштів.

Проект, як і будь-яка фінансова операція, тобто операція, пов'язана з отриманням доходів і (або) здійсненням витрат, породжує грошові потоки (потоки реальних грошей).

Методичними рекомендаціями з оцінки ефективності інвестиційних проектів грошовий потік інвестиційного проекту визначається як залежність від часу грошових надходжень і платежів при реалізації головного, грошовий потік визначається для всього розрахункового періоду.

На кожному етапі життєвого циклу інвестиційного проекту значення грошового потоку характеризується:

— припливом, рівним розміру грошових надходжень (або результатів у вартісному вираженні) в кожен період реалізації інвестиційних проектів;

— відтоком, рівним розміру платежів за кожен з періодів реалізації інвестиційних проектів;

— сальдо (активним балансом, ефектом), рівним різниці між припливом і відтоком.

Основною метою аналізу грошових потоків є визначення величини готівки по напрямками використання та джерелами її надходження. Грошовий приплив в

основному забезпечується за рахунок коштів, що надходять у вигляді виручки від продажу продукції (товарів, робіт, послуг) і з різних джерел фінансування (в результаті продажу акцій і облігацій, отримання банківських кредитів, позик у сторонніх організацій та цільового фінансування, амортизації, використання прибутку). [47]

4.8.1 Формування грошового потоку від реалізації проекту

Чистий дисконтований дохід (NPV, Net Present Value) – це різниця між надходженнями та інвестиціями, отриманими за весь період реалізації проекту.

Таблиця 4.13. Формування грошового потоку від реалізації проекту

№	*Показник*	*Значення по роках*						*Разом*
		2018	*2019*	*2020*	*2021*	*2022*	*2023*	
1.	*Надходження від проекту (виручка від реалізації продукції, послуг – див. п. 7.2) (D)*	130000	130000	130000	130000	260000	260000	
2.	*Загальні витрати (див. п. 7.5) (I), в тому числі*	156000	29000	29000	29000	29000	29000	145000
3.	*Грошовий потік (З = 1 – 2) (CF)*	-26000	101000	101000	101000	231000	231000	739000
4.	*Акумульований грошовий потік (ACF)*	21000	62175	81020	85005	97441	*98065*	-

Висновки до розділу 4

Розроблений програмний продукт є конкурентоздатним на ринку. Основна інвестиційна привабливість проекту у його унікальності та фактично відсутності реальних конкурентів, а також можливості легко вийти на інші, міжнародні ринки з мінімумом витрат. Продукт використовує бізнес-модель продажу одноразової ліцензії та додаткової платної підтримки. Потенційними користувачами програмної

системи є керівники та виконавці завдань у сфері розробки програмного забезпечення що наразі використовують платформу Sharepoint.

Цінність продукту для користувачів полягає у полегшенні можливості відслідковування прогресу виконання того чи іншого проекту та спрощення взаємодії всередині команди.

ВИСНОВКИ

Магістерська робота присвячена розробці інструментів для інтеграції репозиторіїв системи керування версіями з внутрішніми даними системи у вигляді компоненту до існуючої системи Microsoft Office 365. У процесі виконання дипломної роботи проведено аналіз існуючого програмного забезпечення керування версіями програмного коду.

На основі аналізу різних типів систем контролю версій зрозуміло, що децентралізовані системи забезпечують найкращу підтримку керування базою програмного коду, оскільки дозволяють співпрацювати з різними групами людей, застосовуючи різні підходи в межах одного проекту одночасно, а також потребують доступу до мережі тільки для того щоб ділитися наборами змін з людьми.

Проте хмарний пакет послуг Office 365 не містить вбудованих програмних інструментів для інтеграції сучасних систем хостингу репозиторіїв СКВ з внутрішніми даними системи.

Microsoft Sharepoint, що входить до складу Office 365 надає базові можливості роботи з документами, в тому числі файлами програмного коду, але також не може бути використаний в якості системи контролю версій. Розроблений програмний продукт дозволяє вирішити цю проблему.

У даній роботі визначено чіткі вимоги до системи та її функцій. Обґрунтовано вибір використаної клієнт-серверної архітектури та програмних засобів (VueJS, Typescript, IDE Webstorm), обраних для реалізації системи.

Програмний продукт забезпечує наступні можливості:

— можливість роботи з віддаленим репозиторіями програмного коду;
— підтримка різних систем контролю версій (git, tfs, svn);

— використання прошарку, що дозволить абстрагуватися від реалізації REST-API окремої СКВ.

Проект був розроблений у вигляді веб-застосунку SharePoint WebParts, із застосуванням технології односторінкових застосунків, що надає інтерфейс для керування завданнями на підприємстві. Розроблена система включає в себе підсистеми авторизації та компоненти для взаємодії із зовнішніми програмними інтерфейсами віддалених систем контролю версій.

У роботі запропонована архітектура модулю, яка задовольняє поставленій задачі, побудована об'єктна модель задачі організації взаємодії команди при розробці програмного продукту, представлено UML-діаграми бізнес-процесів всередині команди.

Розроблений програмний продукт є конкурентоздатним на ринку. Основна інвестиційна привабливість проекту у його унікальності та фактично відсутності реальних конкурентів, а також можливості легко вийти на інші, міжнародні ринки з мінімумом витрат. Продукт використовує бізнес-модель продажу одноразової ліцензії та додаткової платної підтримки. Потенційними користувачами програмної системи є керівники та виконавці завдань у сфері розробки складних інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Straub B., Chacon S. Pro Git 2nd ed. Edition. APress, 2014. 528 p.
2. Swicegood T. Pragmatic Version Control Using Git. Pragmatic Bookshelf, 2008. 184 p.
3. Соммервилл И. Инженерия программного обеспечения. «Вильямс», 2002. 624 p.
4. Hodson R. Ry's Git Tutorial. RyPress, 2014. 198 p.
5. Silverman R.E. Git Pocket Guide: A Working Introduction. O'Reilly Media, 2013. 234 p.
6. Pilato M., Collins-Sussman B. Version Control with Subversion: Next Generation Open Source Version Control. O'Reilly Media, 2008. 432 p.
7. Version Control Systems Popularity in 2016: [електронний ресурс] Режим доступу: https://rhodecode.com/insights/version-control-systems-2016 (дата звернення: 11.12.2018).
8. Длугунович Н. Особливості організації віддалених комунікацій при розробці програмного забезпечення // Матеріали конференції “INFORMATION, COMMUNICATION, SOCIETY” (ICS-2016). Львів. 2016.
9. В.І. Я., "Аналіз сучасного стану інформаційних технологій та систем в індустрії туризму," // Економічні науки, No. 53, 2017. pp. 56-60.
10. Office 365 Business преміум: [електронний ресурс] Режим доступу:

https://products.office.com/uk-ua/business/office-365-business-premium (дата звернення: 11.12.2018).

11. ПЗ для спільної роботи та групового чату – Microsoft Teams: [електронний ресурс] Режим доступу: https://products.office.com/uk-ua/microsoft-teams/group-chat-software (дата звернення: 10.12.2018).
12. Microsoft Office 365 [Електроний ресурс] //Вікіпедія: [електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Office_365 (дата звернення: 4.12.2018).
13. JSON: [електронний ресурс] [2018]. Режим доступу: https://uk.wikipedia.org/wiki/JSON
14. Introducing JSON: [електронний ресурс] [2018]. Режим доступу: https://www.json.org/
15. Bassett L. Introduction to JavaScript Object Notation. O'Reilly Media, 2015. 126 p.
16. Общие сведения о списках: [електронний ресурс] Режим доступу: https://support.office.com/ru-ru/article/Общие-сведения-о-списках-0a1c3ace-def0-44af-b225-cfa8d92c52d7
17. Withee R. SharePoint 2016 For Dummies. For Dummies, 384.
18. Майкл Ноэл К.С. Microsoft SharePoint 2010. Полное руководство. Вильямс, 2012. 880 p.
19. Perran , Perran S., та Mason. Beginning SharePoint 2013: Building Business Solutions. Wrox, 2013. 672 p.
20. Alirezaei R., Bishop D., та Bleeker T. Real World SharePoint 2010: Indispensable Experiences from 22 MVPs 1st Edition. Wrox, 2010. 816 p.

21. Londer O., Coventry P. Microsoft SharePoint 2013 Step by Step 1st Edition. Microsoft Press, 688.

22. Copes F. The vue handbook. 2018. 121 p.

23. Kocherhin O. Mastering Vue.js. 2018. 362 p.

24. Руби С. Т.Д..Х.Д.Х. Гибкая разработка веб-приложений в среде Rails. 4-е изд. СПБ: Питер, 2012. 464 с.

25. Директивы: [електронний ресурс] [2018]. Режим доступу: https://ru.vuejs.org/v2/guide/syntax.html#Директивы (дата звернення: 4.12.2018).

26. Konshin K. Next.js Quick Start Guide. Packt Publishing, 2018. 164 p.

27. Руководство по серверному рендерингу Vue.js 2018. URL: https://ssr.vuejs.org/ru/#нужен-ли-вам-ssr

28. Gore A. Full-Stack Vue.js 2 and Laravel 5. Packt Publishing, 2017. 378 p.

29. Maniatis K., Kyriakidis A., та You. The Majesty of Vue.js 2. 2017. 333 p.

30. SSR. Отрисовка на стороне сервера: [електронний ресурс] [2018]. Режим доступу: https://ru.vuejs.org/v2/guide/ssr.html#Nuxt-js (дата звернення: 12.04.2018).

31. WebStorm: [електронний ресурс] Режим доступу: https://ru.wikipedia.org/wiki/WebStorm (дата звернення: 08.11.2018).

32. Файн Я., Моисеев А. Angular и TypeScript. Сайтостроение для профессионалов. Питер, 2018. 464 p.

33. METANIT - Что такое TypeScript: [електронний ресурс] [2018]. Режим доступу: https://metanit.com/web/typescript/1.1.php (дата звернення: 9.11.2018).

34. TypeScript: [електронний ресурс] Режим доступу: https:// uk.wikipedia.org/wiki/TypeScript (дата звернення: 08.12.2018).

35. Nance C. TypeScript Essentials. Packt Publishing, 2014. 182 p.

36. Crockford D. How JavaScript Works. 2018. 279 p.

37. Экземпляр Vue: [електронний ресурс] [2018]. Режим доступу: https:// ru.vuejs.org/v2/guide/instance.html (дата звернення: 5.11.2018).

38. Требования к оборудованию и программному обеспечению для SharePoint Server 2016: [електронний ресурс] Режим доступу: https:// docs.microsoft.com/ru-ru/sharepoint/install/hardware-and-software-requirements (дата звернення: 5.12.2018).

39. Дейв , Санни Б., та Джеймс М. Геймшторминг. Игры, в которые играет бизнес. 2012. 288 p.

40. Владимир Репин В.Е. Процессный подход к управлению. Моделирование бизнес-процессов. Манн, Иванов и Фербер, 2012. 544 p.

41. Александр Остервальдер И.П.Г.Б.А.С. Разработка ценностных предложений. Как создавать товары и услуги, которые захотят купить потребители. Альпина Паблишер, 2018. 312 p.

42. Еганян А. Инвестиции в инфраструктуру. Деньги, проекты, интересы. ГЧП, концессии, проектное финансирование. Альпина Паблишер, 2018. 720 p.

43. Пер Дженстер Д.Х. Анализ сильных и слабых сторон компании. Определение стратегических возможностей. Диалектика, 2016. 368 p.

44. MBA для "чайников". Вильямс, 2006. 352 p.

45. Кеворков В.В. К.Д.В. Практикум по маркетингу. 4-е издание. Проспект, 2013. 724 р.

46. Казинцев А. Шесть Сигм в России: методика снижения потерь-дефектов-издержек. 2009. 368 р.

47. Киселёва О.В. М.Ф.С. Инвестиционный анализ. Учебное пособие. КНОРУС, 2010. 208 р.

48. Scott E. SPA Design and Architecture: Understanding Single Page Web Applications. Manning Publications, 2015. P. 275.

49. S.r A. Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. New York: John Wiley & Sons, 2002. 402 с.

50. Крокфорд Д. JavaScript. Сильные стороны. Питер: СПБ, 2013. 176 р.

51. Д Ф. JavaScript. Подробное руководство. СПб: Символ-Плюс, 2008. 992 р.

52. К Д. Введення в системи баз даних — 8-ме вид. М.: «Вильямс», 2006. 1328 р.

ДОДАТОК А

Система керування базою програмного коду на платформі Office 365

Апробація

УКР.НТУУ “КПІ” ім. І. Сікорського. ТМ31131_18М

Аркушів 4

2018

УДК 621.43.056:632.15

Магістрант 5 курсу, гр. ТМ-71мп Литвиненко Д.С.
Доц., к.т.н. Тихоход В.О.

СИСТЕМА КЕРУВАННЯ БАЗОЮ ПРОГРАМНОГО КОДУ НА ПЛАТФОРМІ OFFICE 365

Розробка складних інформаційних систем вимагає узгодженої роботи цілої групи програмістів. Проблеми організації взаємодії при розробці програмного продукту великою командою розробників зазвичай вирішується використанням системи керування версіями програмного коду (Version Control System, VCS) [1]. Така система дозволяє керувати поступовими змінами внесеними в електронні документи та відміняти ці зміни у разі необхідності.

В той же час ця система керування версіями не існує сама по собі, а тісно пов'язана із іншими внутрішніми інструментами та системами, що використовуються в процесі розробки та підтримки програмних продуктів. До них відносять системи документації, керування задачами (такими як Redmine, Jira), системами неперервної інтеграції (Jenkins) та внутрішніми продуктами компанії.

Наразі на кафедрі ведеться розробка компонентів для єдиної системи на основі хмарного офісного пакету послуг Microsoft Office 365, що забезпечить користувачам доступ до документів і даних через браузер з будь-якого пристрою з можливістю виходу в Інтернет. Пакет послуг Microsoft Office 365 включає в себе SharePoint Online для створення веб-сайту організації і внутрішніх соціальних мереж для спілкування та взаємодії співробітників [2]. Сайти, які створюються на платформі SharePoint, можна застосовувати як сховища інформації про розробки окремих лабораторій та документів пов'язаних із цими розробками, а також використовувати для виконання веб-застосунків, таких як вікі і блоги.

Наразі в системі відсутня можливість інтеграції із віддаленими репозиторіями систем керування версій.

Для вирішення цієї проблеми пропонується розробити систему керування базою програмного коду у вигляді веб-застосунку (SharePoint WebParts). Перевагою використання WebParts є можливість розробки системи без використання центрального серверу, із застосуванням технології односторінкових застосунків (single-page application).

Розроблена система дозволить пов'язати конкретні проекти в системі Microsoft SharePoint з репозиторієм коду та підвищити рівень і якість взаємодії розробників у команді, зменшити питомі витрати на синхронізацію цих змін з іншими, суміжними проектами.

Таким чином, можна виділити основні функції системи керування базою програмного коду на платформі office 365:

- Можливість роботи з віддаленим репозиторіями програмного коду
- Підтримка різних систем контролю версій (git, tfs, svn)
- Використання прошарку що дозволить абстрагуватися від реалізації окремого репозиторію або системи контролю версій

Характерною особливістю системи є можливість вибору різних централізованих сховищ програмного коду(github, gitlab або власного серверу) як джерела даних.

Перелік посилань:

1. Pro Git 2nd ed. Edition / S. Chacon, B. Straub — Apress, 2014. - 528 p. ISBN 978-1484200773.

2. Microsoft_Office_365 [Електроннии□ ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/Microsoft_Office_365.

ДОДАТОК Б

Система керування базою програмного коду на платформі Office 365